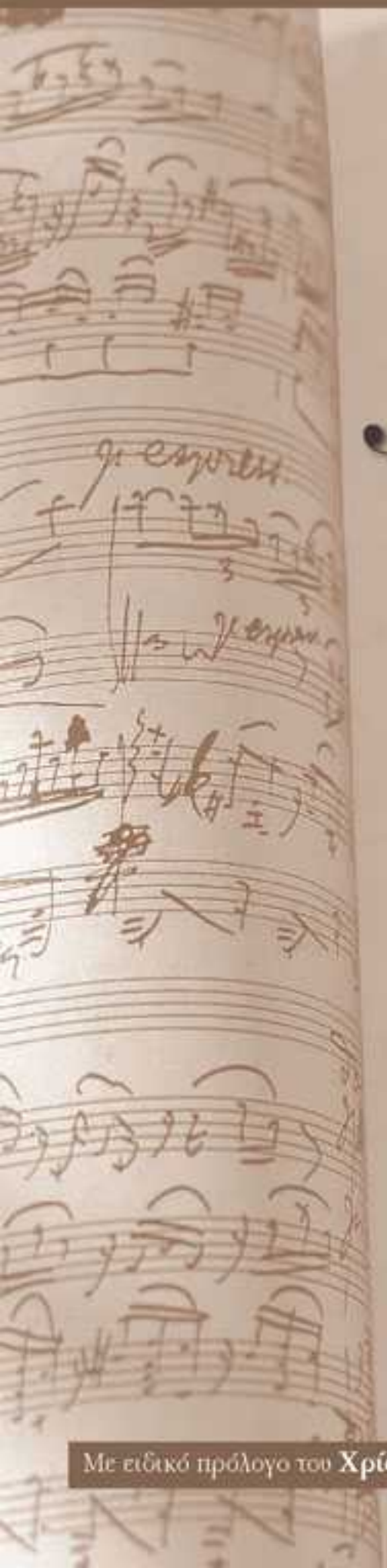
ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ



Αλγόριθμοι

Sanjoy Dasgupta
Christos Papadimitriou
Umesh Vazirani

Επιστημονική επιμέλεια ελληνικής έκδοσης:
Γιώργος Στεφανίδης
Τμ. Εφαρμοσμένης Πληροφορικής, Πανεπ. Μακεδονίας

Με ειδικό πρόλογο του **Χρίστου Παπαδημητρίου** για την ελληνική έκδοση

ΚΛΕΙΔΑΡΙΘΜΟΣ

Περιεχόμενα

Κεφάλαιο 6

Δυναμικός προγραμματισμός

Στα προηγούμενα κεφάλαια έχουμε δει μερικές κομψές αρχές σχεδίασης — όπως διαίρει και βασίλευε, εξερεύνηση γραφημάτων, και άπληστη επιλογή — οι οποίες παρέχουν καθοριστικούς αλγορίθμους για μια ποικιλία σημαντικών υπολογιστικών εργασιών. Το μειονέκτημα αυτών των εργαλείων είναι ότι μπορούν να χρησιμοποιηθούν μόνο για πολύ ειδικούς τύπους προβλημάτων. Τώρα θα στραφούμε στα δύο βασικά εργαλεία της τέχνης των αλγορίθμων, το *δυναμικό προγραμματισμό* και το *γραμμικό προγραμματισμό*, τεχνικές με ευρύτατο πεδίο εφαρμογών οι οποίες μπορούν να χρησιμοποιηθούν όταν οι πιο εξειδικευμένες μέθοδοι αποτυγχάνουν. Όπως όμως είναι αναμενόμενο, αυτή η γενικότητα συνοδεύεται συχνά από ένα κόστος ως προς την αποδοτικότητα.

6.1 Και πάλι για τις συντομότερες διαδρομές σε DAG

Στο τέλος της μελέτης μας για τις συντομότερες διαδρομές (Κεφάλαιο 4) παρατηρήσαμε ότι το πρόβλημα είναι ιδιαίτερα εύκολο σε προσανατολισμένα άκυκλα γραφήματα (directed acyclic graphs, DAG). Ας κάνουμε μια ανασκόπηση αυτής της περίπτωσης, επειδή βρίσκεται στην καρδιά του δυναμικού προγραμματισμού.

Το ειδικό χαρακτηριστικό το οποίο διακρίνει ένα DAG είναι ότι οι κόμβοι του μπορούν να *γραμμικοποιηθούν*· δηλαδή μπορούν να διευθετηθούν σε μια γραμμή έτσι ώστε όλες οι ακμές να κατευθύνονται από τα αριστερά προς τα δεξιά (Εικόνα 6.1). Για να δείτε γιατί αυτό το χαρακτηριστικό βοηθά στην περίπτωση των συντομότερων διαδρομών, υποθέστε ότι θέλουμε να υπολογίσουμε τις αποστάσεις από τον κόμβο S προς τους άλλους κόμβους. Για να έχουμε ένα χειροπιαστό παράδειγμα, ας εστιάσουμε στον κόμβο D. Ο μόνος τρόπος για να φθάσουμε σε αυτόν είναι μέσω των προκατόχων του, B ή C· επομένως, για να βρούμε τη συντομότερη διαδρομή μέχρι το D, πρέπει μόνο να συγκρίνουμε αυτά τα δύο δρομολόγια:

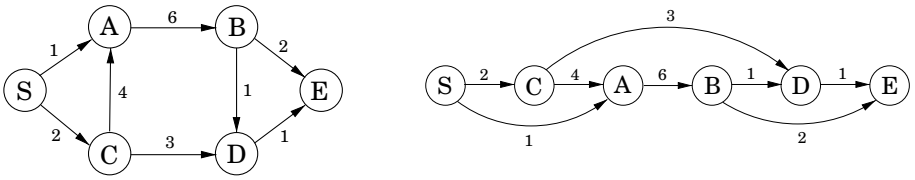
$$\texttt{dist}(D) = \min\{\texttt{dist}(B) + 1, \texttt{dist}(C) + 3\}.$$

Μια παρόμοια σχέση μπορεί να γραφτεί για κάθε κόμβο. Αν υπολογίσουμε αυτές τις τιμές `dist` με σειρά από τα αριστερά προς τα δεξιά στην Εικόνα 6.1, μπορούμε πάντοτε να είμαστε σίγουροι ότι όταν φθάσουμε στον κόμβο v θα έχουμε ήδη όλες τις πληροφο-

ρίες τις οποίες χρειαζόμαστε για να υπολογίσουμε την απόσταση dist(v). Επομένως είμαστε σε θέση να υπολογίσουμε όλες τις αποστάσεις με ένα πέρασμα:

```
ανάθεση αρχικής τιμής ∞ σε όλες τις τιμές dist(·)
dist(s) = 0
for κάθε v ∈ V\{s}, με γραμμικοποιημένη σειρά:
    dist(v) = min_{(u,v)∈E} {dist(u) + l(u, v)}
```

Εικόνα 6.1 Ένα DAG και η γραμμικοποίησή του (τοπολογική διάταξη).



Παρατηρήστε ότι αυτός ο αλγόριθμος επιλύει μια συλλογή *υποπροβλημάτων*, {dist(u): $u \in V$}. Αρχίζουμε με το μικρότερο από αυτά, dist(s), καθώς γνωρίζουμε αμέσως ότι η απάντησή του είναι 0. Κατόπιν προχωρούμε με προοδευτικά «μεγαλύτερα» υποπροβλήματα — αποστάσεις από κορυφές οι οποίες είναι όλο και πιο μακριά στη γραμμικοποίηση — όπου θεωρούμε ένα υποπρόβλημα «μεγάλο» αν πρέπει να λύσουμε πολλά άλλα υποπροβλήματα πριν φθάσουμε σε αυτό.

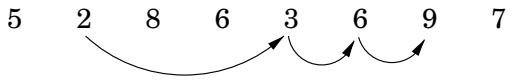
Η τεχνική αυτή είναι πολύ γενική. Σε κάθε κόμβο υπολογίζουμε κάποια συνάρτηση των τιμών των προκατόχων του κόμβου. Τυχαίνει η συγκεκριμένη συνάρτησή μας να είναι ένα ελάχιστο αθροισμάτων, αλλά θα μπορούσε εξίσου καλά να είναι ένα *μέγιστο*, οπότε θα μπορούσαμε να πάρουμε τις *μεγαλύτερες* διαδρομές στο DAG. Ή θα μπορούσαμε να χρησιμοποιήσουμε μέσα στις αγκύλες ένα γινόμενο αντί για άθροισμα, οπότε σε αυτή την περίπτωση θα καταλήγαμε να υπολογίζουμε τη διαδρομή η οποία έχει το μικρότερο γινόμενο μηκών των ακμών.

Ο *δυναμικός προγραμματισμός* (dynamic programming) είναι ένα πολύ ισχυρό αλγοριθμικό υπόδειγμα όπου για να λύσουμε ένα πρόβλημα αναγνωρίζουμε μια συλλογή υποπροβλημάτων και τα αντιμετωπίζουμε ένα προς ένα· ξεκινώντας από το μικρότερο, χρησιμοποιούμε τις απαντήσεις στα μικρά προβλήματα για να υπολογίσουμε τα μεγαλύτερα, μέχρι να τα λύσουμε όλα. Στο δυναμικό προγραμματισμό δεν μας δίνεται ένα DAG· το DAG είναι *υπονοούμενο.* Οι κόμβοι του είναι τα υποπροβλήματα τα οποία ορίζουμε, και οι ακμές του είναι οι εξαρτήσεις ανάμεσα στα υποπροβλήματα: αν για να λύσουμε το πρόβλημα B χρειαζόμαστε την απάντηση στο υποπρόβλημα A, τότε υπάρχει μια (ιδεατή) ακμή από το A στο B. Σε αυτή την περίπτωση το A θεωρείται μικρότερο υποπρόβλημα από το B — και πάντοτε θα είναι μικρότερο, κατά την προφανή έννοια.

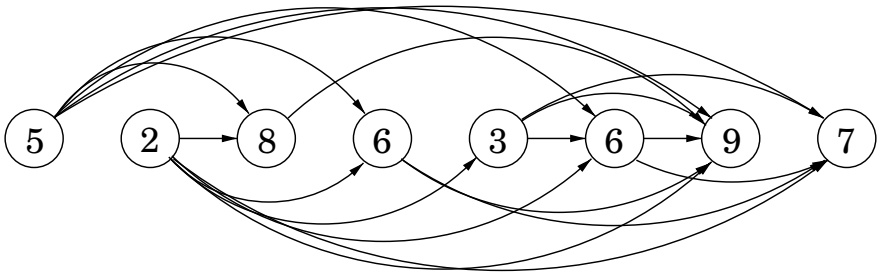
Ήρθε όμως η ώρα να δούμε ένα παράδειγμα.

6.2 Μέγιστες αύξουσες υπακολουθίες

Στο πρόβλημα της *μέγιστης αύξουσας υπακολουθίας* (longest increasing subsequence), η είσοδος είναι μια ακολουθία αριθμών $a_1,..., a_n$. *Υπακολουθία* (subsequence) είναι ένα υποσύνολο αυτών των αριθμών που έχουν ληφθεί με τη σειρά, της μορφής $a_{i_1}, a_{i_2}, ..., a_{i_k}$, όπου $1 \le i_1 < i_2 < \cdots < i_k \le n$, και *αύξουσα* υπακολουθία είναι αυτή στην οποία οι αριθμοί γίνονται γνησίως μεγαλύτεροι. Το εγχείρημα είναι να βρούμε την αύξουσα υπακολουθία με το μέγιστο μήκος. Για παράδειγμα, η μέγιστη αύξουσα υπακολουθία για το 5, 2, 8, 6, 3, 6, 9, 7 είναι η 2, 3, 6, 9:



Εικόνα 6.2 Το DAG των αυξουσών υπακολουθιών.



Σε αυτό το παράδειγμα τα βέλη υποδηλώνουν μεταβάσεις ανάμεσα σε διαδοχικά στοιχεία της βέλτιστης λύσης. Πιο γενικά, για να καταλάβετε καλύτερα το περιβάλλον της λύσης, ας δημιουργήσουμε ένα γράφημα με *όλες* τις επιτρεπτές μεταβάσεις: δημιουργούμε έναν κόμβο i για κάθε στοιχείο a_i, και προσθέτουμε προσανατολισμένες ακμές (i, j) οποτεδήποτε είναι δυνατό τα a_i και a_j να είναι διαδοχικά στοιχεία σε μια αύξουσα υπακολουθία, δηλαδή κάθε φορά που $i < j$ και $a_i < a_j$ (Εικόνα 6.2).

Παρατηρήστε ότι (1) αυτό το γράφημα $G = (V, E)$ είναι ένα DAG, αφού όλες οι ακμές (i, j) έχουν $i < j$, και (2) ότι υπάρχει μια αντιστοιχία ένα προς ένα ανάμεσα στις αύξουσες υπακολουθίες και στις διαδρομές αυτού του DAG. Κατά συνέπεια, σκοπός μας είναι απλώς να βρούμε τη μεγαλύτερη διαδρομή στο DAG!

Ορίστε ο αλγόριθμος:

```
for j = 1, 2,..., n:
  L( j ) = 1 + max{L(i): (i, j ) ∈ E }
return max_j L( j )
```

$L(j)$ είναι το μήκος της μεγαλύτερης διαδρομής — η μέγιστη αύξουσα υπακολουθία — που τελειώνει στον κόμβο j (συν 1, αφού για να είμαστε αυστηροί πρέπει να μετρήσουμε τους κόμβους της διαδρομής, και όχι τις ακμές). Ακολουθώντας την ίδια συλλογιστική την οποία χρησιμοποιήσαμε με τις συντομότερες διαδρομές, βλέπουμε ότι οποιαδήποτε διαδρομή προς τον κόμβο j πρέπει να διέρχεται μέσω κάποιου από τους προκατόχους του, και επομένως το $L(j)$ είναι 1 συν τη μέγιστη τιμή $L(\cdot)$ αυτών των προκατόχων. Αν δεν υπάρχουν εισερχόμενες ακμές στον j, παίρνουμε το μέγιστο για το κενό σύνολο, που είναι μηδέν. Και η τελική απάντηση είναι το *μεγαλύτερο* $L(j)$, αφού επιτρέπεται οποιαδήποτε τελική θέση.

Αυτό είναι δυναμικός προγραμματισμός. Για να λύσουμε το αρχικό μας πρόβλημα, έχουμε ορίσει μια συλλογή υποπροβλημάτων $\{L(j)\colon 1 \le j \le n\}$ με την ακόλουθη καίρια ιδιότητα η οποία επιτρέπει την επίλυσή τους με ένα πέρασμα:

(*) *Υπάρχει μια διάταξη των υποπροβλημάτων, και μια σχέση η οποία δείχνει πώς θα λυθεί ένα υποπρόβλημα με δεδομένες τις απαντήσεις σε «μικρότερα» υποπροβλήματα, δηλαδή υποπροβλήματα τα οποία εμφανίστηκαν νωρίτερα στη διάταξη.*

Στη δική μας περίπτωση, κάθε υποπρόβλημα λύνεται με τη χρήση της σχέσης

$$L(j) = 1 + \max\{L(i)\colon (i, j) \in E\},$$

μια παράσταση η οποία περιλαμβάνει μόνο μικρότερα υποπροβλήματα. Πόσο χρόνο χρειάζεται αυτό το βήμα; Απαιτεί να είναι γνωστοί οι προκάτοχοι του j· για τη δουλειά αυτή είναι χρήσιμη η λίστα γειτνίασης του αντίστροφου γραφήματος G^R, που μπορεί να κατασκευαστεί σε γραμμικό χρόνο (θυμηθείτε την Άσκηση 3.5). Επομένως, ο υπολογισμός του $L(j)$ χρειάζεται χρόνο ο οποίος είναι ανάλογος του βαθμού εισόδου (indegree) του j, άρα έχουμε συνολικό χρόνο εκτέλεσης γραμμικό ως προς $|E|$. Αυτός ο χρόνος είναι το πολύ $O(n^2)$, όπου η μέγιστη τιμή παρατηρείται όταν η συστοιχία εισόδου είναι ταξινομημένη κατ' αύξουσα σειρά. Κατά συνέπεια, η λύση δυναμικού προγραμματισμού είναι και απλή και αποδοτική.

Υπάρχει ένα τελευταίο ζήτημα το οποίο πρέπει να διασαφηνιστεί: οι τιμές L μας λένε μόνο το *μήκος* της βέλτιστης υπακολουθίας, οπότε πώς θα ανακτήσουμε την ίδια την υπακολουθία; Αυτό μπορούμε να το καταφέρουμε εύκολα με τον ίδιο λογιστικό μηχανισμό τον οποίο χρησιμοποιήσαμε στο Κεφάλαιο 4 για τις συντομότερες διαδρομές. Κατά τον υπολογισμό του $L(j)$ θα πρέπει να σημειώνουμε το $\texttt{prev}(j)$, τον προτελευταίο κόμβο της μεγαλύτερης διαδρομής μέχρι τον κόμβο j. Έτσι μπορούμε να ανακατασκευάσουμε τη βέλτιστη υπακολουθία ακολουθώντας προς τα πίσω αυτούς τους δείκτες.

6.3 Διορθωτική απόσταση

Όταν ένας ελεγκτής ορθογραφίας συναντά μια πιθανή ανορθογραφία, εξετάζει το λεξικό του για άλλες λέξεις που είναι κοντά στην ανορθόγραφη. Ποια είναι η κατάλληλη έννοια της εγγύτητας σε αυτή την περίπτωση;

Ένα φυσικό μέτρο της απόστασης ανάμεσα σε δύο συμβολοσειρές είναι η έκταση στην οποία μπορούν να *στοιχηθούν* (aligned), δηλαδή να συνταιριαστούν. Τεχνικά, η στοίχιση είναι απλώς ένας τρόπος γραφής των συμβολοσειρών έτσι ώστε η μία να είναι επάνω από την άλλη. Για παράδειγμα, ακολουθούν δύο πιθανές στοιχίσεις των λέξεων SNOWY και SUNNY.

S	–	N	O	W	Y		–	S	N	O	W	–	Y
S	U	N	N	–	Y		S	U	N	–	–	N	Y
Κόστος: 3							Κόστος: 5						

Το σύμβολο «–» υποδηλώνει ένα «κενό»· σε κάθε συμβολοσειρά μπορεί να τοποθετηθεί οποιοδήποτε πλήθος τέτοιων συμβόλων. Το *κόστος* μιας στοίχισης είναι το πλήθος των στηλών στις οποίες διαφέρουν τα γράμματα. Και η *διορθωτική απόσταση* (edit distance) μεταξύ των δύο συμβολοσειρών είναι το κόστος των καλύτερων δυνατών στοιχίσεών τους. Βλέπετε γιατί δεν υπάρχει καλύτερη στοίχιση των λέξεων SNOWY και SUNNY από αυτή που φαίνεται εδώ με κόστος 3;

Η διορθωτική απόσταση ονομάζεται έτσι επειδή μπορεί να θεωρηθεί ως το ελάχιστο πλήθος *διορθώσεων* — εισαγωγών, διαγραφών, και αντικαταστάσεων χαρακτήρων — οι οποίες είναι απαραίτητες για το μετασχηματισμό της πρώτης συμβολοσειράς στη δεύτερη. Για παράδειγμα, η στοίχιση που φαίνεται στα αριστερά αντιστοιχεί σε τρεις διορθώσεις: την εισαγωγή του U, την αντικατάσταση O → N, και τη διαγραφή του W.

Γενικά, υπάρχουν τόσο πολλές δυνατές στοιχίσεις ανάμεσα σε δύο συμβολοσειρές ώστε θα ήταν τρομερά αναποτελεσματική η εξέταση όλων αυτών για την εύρεση της καλύτερης. Έτσι στρεφόμαστε στο δυναμικό προγραμματισμό.

Μια λύση δυναμικού προγραμματισμού

Όταν λύνετε ένα πρόβλημα με δυναμικό προγραμματισμό, το πιο κρίσιμο ερώτημα είναι *Ποια είναι τα υποπροβλήματα;* Εφ' όσον αυτά επιλέγονται έτσι ώστε να έχουν την ιδιότητα (*) της σελίδας 208, είναι εύκολη υπόθεση η συγγραφή του αλγορίθμου: επαναληπτικά επιλύουμε το ένα υποπρόβλημα μετά το άλλο, κατά σειρά αυξανόμενου μεγέθους.

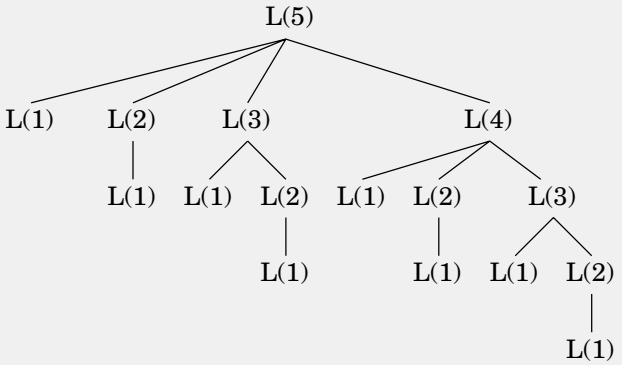
Αναδρομή; Όχι, ευχαριστώ.

Επιστρέφουμε στην περιγραφή μας για τις μέγιστες αύξουσες υπακολουθίες: ο τύπος για το $L(j)$ παραπέμπει σε έναν εναλλακτικό, αναδρομικό αλγόριθμο. Αυτός ο αλγόριθμος δε θα ήταν ακόμη πιο απλός;

Στην πραγματικότητα, η αναδρομή είναι μια πολύ κακή ιδέα: η διαδικασία η οποία προκύπτει θα απαιτούσε εκθετικό χρόνο! Για να δείτε γιατί, ας υποθέσουμε ότι το DAG περιέχει ακμές (i, j) για *όλα* τα $i < j$ — δηλαδή η δεδομένη ακολουθία των αριθμών $a_1, a_2, \ldots, a_n$ είναι ταξινομημένη. Σε αυτή την περίπτωση, ο τύπος για το υποπρόβλημα $L(j)$ γίνεται

$$L(j) = 1 + \max\{L(1), L(2), \ldots, L(j-1)\}.$$

Η εικόνα που ακολουθεί αποκαλύπτει την αναδρομή για το $L(5)$. Παρατηρήστε ότι επιλύονται συνεχώς τα ίδια υποπροβλήματα!



Για το $L(n)$ αυτό το δένδρο έχει εκθετικό πλήθος κόμβων (μπορείτε να φράξετε αυτό το πλήθος;), οπότε μια αναδρομική λύση είναι καταστροφική.

Γιατί τότε η αναδρομή αποδίδει τόσο καλά με την τεχνική διαίρει και βασίλευε; Το καίριο ζήτημα είναι ότι στην προσέγγιση διαίρει και βασίλευε ένα πρόβλημα εκφράζεται συναρτήσει υποπροβλημάτων τα οποία είναι *σημαντικά μικρότερα*, για παράδειγμα μισού μεγέθους. Για παράδειγμα, η διαδικασία `mergsort` ταξινομεί μια συστοιχία μεγέθους n μέσω αναδρομικής ταξινόμησης δύο υποσυστοιχιών μεγέθους $n/2$. Λόγω αυτής της απότομης μείωσης στο μέγεθος του προβλήματος, το πλήρες δένδρο της αναδρομής έχει μόνο λογαριθμικό βάθος και πολυωνυμικό πλήθος κόμβων.

Αντιθέτως, σε μια τυπική διατύπωση δυναμικού προγραμματισμού το πρόβλημα μειώνεται σε υποπροβλήματα τα οποία είναι ελάχιστα μικρότερα — για παράδειγμα, το $L(j)$ βασίζεται στο $L(j-1)$. Κατά συνέπεια, το πλήρες δένδρο αναδρομής έχει πολυωνυμικό βάθος και εκθετικό πλήθος κόμβων. Ωστόσο, αποδεικνύεται ότι οι περισσότεροι από αυτούς τους κόμβους είναι επαναλήψεις, και ότι δεν υπάρχουν πάρα πολλά *διαφορετικά* υποπροβλήματα μεταξύ αυτών. Έτσι η αποδοτικότητα επιτυγχάνεται με τη ρητή απαρίθμηση των διαφορετικών υποπροβλημάτων και την επίλυσή τους με τη σωστή σειρά.

Προγραμματισμός;

Η προέλευση του όρου *δυναμικός προγραμματισμός* πολύ λίγο έχει να κάνει με τη συγγραφή κώδικα. Δημιουργήθηκε για πρώτη φορά από τον Richard Bellman στη δεκαετία του 1950, μια εποχή όπου ο προγραμματισμός υπολογιστών ήταν μια απόκρυφη δραστηριότητα την οποία ασκούσαν τόσο λίγοι άνθρωποι που δεν άξιζε ούτε καν να αναφερθούν. Εκείνη την εποχή ο όρος προγραμματισμός σήμαινε «σχεδιασμός ενεργειών» (planning), και ο όρος «δυναμικός προγραμματισμός» χρησιμοποιούνταν για το βέλτιστο σχεδιασμό διεργασιών πολλών σταδίων. Το DAG της Εικόνας 6.2 μπορεί να θεωρηθεί ότι περιγράφει τους δυνατούς τρόπους με τους οποίους μπορεί να εξελιχθεί μια διεργασία: κάθε κόμβος αναπαριστάνει μια κατάσταση, ο αριστερότερος κόμβος είναι το σημείο εκκίνησης, και οι ακμές που εγκαταλείπουν μια κατάσταση αναπαριστάνουν δυνατές ενέργειες, οι οποίες οδηγούν σε διαφορετικές καταστάσεις στην επόμενη μονάδα του χρόνου.

Η ετυμολογία του όρου *γραμμικός προγραμματισμός*, που είναι το αντικείμενο του Κεφαλαίου 7, είναι παρόμοια.

Στόχος μας είναι η εύρεση της διορθωτικής απόστασης ανάμεσα στις δύο συμβολοσειρές $x[1...m]$ και $y[1...n]$. Τι είναι όμως ένα καλό υποπρόβλημα; Θα πρέπει σε κάποιο βαθμό να προχωρά προς τη λύση του συνολικού προβλήματος· τι θα λέγατε λοιπόν να εξετάσουμε τη διορθωτική απόσταση ανάμεσα σε κάποιο *πρόθεμα* (prefix) της πρώτης συμβολοσειράς $x[1...i]$, και κάποιο *πρόθεμα* της δεύτερης, $y[1...n]$; Ας ονομάσουμε αυτό το υποπρόβλημα $E(i, j)$ (δείτε την Εικόνα 6.3). Έτσι, ο τελικός μας στόχος είναι να υπολογίσουμε το $E(m, n)$.

Για να λειτουργήσει αυτή η προσέγγιση, πρέπει με κάποιον τρόπο να εκφράσουμε το $E(i, j)$ συναρτήσει μικρότερων υποπροβλημάτων. Ας δούμε — τι γνωρίζουμε σχετικά με την καλύτερη στοίχιση ανάμεσα στις συμβολοσειρές $x[1...i]$ και $y[1...j]$; Λοιπόν, γνωρίζουμε ότι η δεξιότερη στήλη μπορεί να είναι ένα από τα ακόλουθα τρία πράγματα:

$$\begin{matrix} x[i] \\ - \end{matrix} \quad \text{ή} \quad \begin{matrix} - \\ y[j] \end{matrix} \quad \text{ή} \quad \begin{matrix} x[i] \\ y[j] \end{matrix}$$

Η πρώτη περίπτωση επιφέρει κόστος 1 για τη συγκεκριμένη στήλη, και απομένει να στοιχίσουμε την $x[1...i-1]$ με την $y[1...j]$. Αλλά αυτό είναι ακριβώς το υποπρόβλημα $E(i-1, j)$! Φαίνεται ότι κάπου φθάνουμε. Στη δεύτερη περίπτωση, επίσης με κόστος 1, θα πρέπει να στοιχίσουμε την $x[1...i]$ με την $y[1...j-1]$. Και πάλι, αυτό είναι ένα άλλο υποπρόβλημα, το $E(i, j-1)$. Και στην τελευταία περίπτωση, με κόστος είτε 1 (αν $x[i] \neq y[j]$) είτε 0 (αν $x[i] = y[j]$), αυτό που απομένει είναι το υποπρόβλημα $E(i-1, j-1)$. Εν ολίγοις, έχουμε εκφράσει το $E(i, j)$ συναρτήσει τριών *μικρότερων* υποπροβλημάτων

$E(i-1, j)$, $E(i, j-1)$, $E(i-1, j-1)$. Δε γνωρίζουμε ποιο από αυτά είναι το σωστό, οπότε πρέπει να τα δοκιμάσουμε όλα και να επιλέξουμε το καλύτερο:

$$E(i, j) = \min\{1 + E(i-1, j), 1 + E(i, j-1), \text{diff}(i, j) + E(i-1, j-1)\}$$

όπου για λόγους ευκολίας το diff(i, j) ορίζεται ίσο με 0 αν $x[i] = y[j]$ και 1 διαφορετικά.

Εικόνα 6.3 Το υποπρόβλημα $E(7, 5)$.

EXPONEN TIAL
POLYNOM IAL

Για παράδειγμα, κατά τον υπολογισμό της διορθωτικής απόστασης μεταξύ των λέξεων EXPONENTIAL και POLYNOMIAL, το υποπρόβλημα $E(4, 3)$ αντιστοιχεί στα προθέματα EXPO και POL. Η δεξιότερη στήλη της καλύτερής τους στοίχισης πρέπει να είναι μία από τις ακόλουθες:

O	ή	–	ή	O
–		L		L

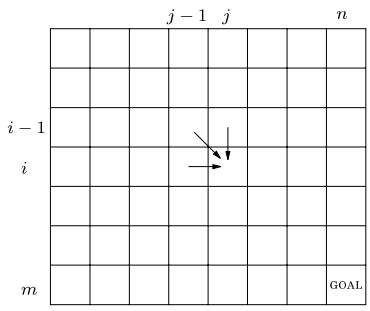
Επομένως, $E(4, 3) = \min\{1 + E(3, 3), 1 + E(4, 2), 1 + E(3,2)\}$.

Οι απαντήσεις σε όλα τα υποπροβλήματα $E(i, j)$ σχηματίζουν ένα διδιάστατο πίνακα, όπως φαίνεται στην Εικόνα 6.4. Με ποια σειρά θα πρέπει να λυθούν αυτά τα υποπροβλήματα; Οποιαδήποτε σειρά είναι καλή, εφόσον ο χειρισμός των υποπροβλημάτων $E(i-1, j)$, $E(i, j-1)$, και $E(i-1, j-1)$ γίνεται *πριν* από το υποπρόβλημα $E(i, j)$. Για παράδειγμα, θα μπορούσαμε να συμπληρώνουμε τον πίνακα κατά μία γραμμή τη φορά, από τη γραμμή στην κορυφή μέχρι την τελευταία γραμμή, ενώ σε κάθε γραμμή να κινούμαστε από τα αριστερά προς τα δεξιά. Ή εναλλακτικά θα μπορούσαμε να τον συμπληρώνουμε στήλη-στήλη. Και οι δύο μέθοδοι εξασφαλίζουν ότι, τη στιγμή που θα υπολογίζαμε μια συγκεκριμένη καταχώριση του πίνακα, όλες οι άλλες καταχωρίσεις που θα χρειαζόμασταν θα ήταν ήδη συμπληρωμένες.

Αφού καθορίστηκαν και τα υποπροβλήματα και η σειρά επίλυσης, έχουμε σχεδόν τελειώσει. Απομένουν απλώς οι «βασικές περιπτώσεις» του δυναμικού προγραμματισμού, τα πάρα πολύ μικρά υποπροβλήματα. Στην παρούσα κατάσταση, αυτά είναι τα $E(0, \cdot)$ και $E(\cdot, 0)$, τα οποία λύνονται και τα δύο εύκολα. Το υποπρόβλημα $E(0, j)$ είναι η διορθωτική απόσταση ανάμεσα στο μηδενικού μήκους πρόθεμα του x, δηλαδή της κενής συμβολοσειράς, και τα πρώτα j γράμματα του y: προφανώς, j. Και παρομοίως, $E(i, 0) = i$.

Εικόνα 6.4 (α) Ο πίνακας των υποπροβλημάτων. Οι καταχωρίσεις $E(i-1, j-1)$, $E(i-1, j)$, και $E(i, j-1)$ είναι αναγκαίες για τη συμπλήρωση του $E(i, j)$. (β) Ο τελικός πίνακας των τιμών που βρέθηκαν με δυναμικό προγραμματισμό.

(α)



(β)

		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	7
L	11	10	9	8	9	8	8	8	8	7	6

Σε αυτό το σημείο, ο αλγόριθμος για τη διορθωτική απόσταση βασικά γράφεται μόνος του.

```
for i = 0, 1, 2,..., m:
    E (i, 0) = i
for j = 1, 2,..., n:
    E(0, j) = j
for i = 1, 2,..., m:
    for j = 1, 2,..., n:
        E(i, j ) = min{E(i − 1, j) + 1, E(i, j − 1) + 1, E(i − 1, j − 1) + diff(i, j)}
return E(m, n)
```

Αυτή η διαδικασία συμπληρώνει τον πίνακα γραμμή-γραμμή, και σε κάθε γραμμή από αριστερά προς τα δεξιά. Κάθε καταχώριση χρειάζεται σταθερό χρόνο για να συμπληρωθεί, οπότε ο συνολικός χρόνος εκτέλεσης είναι απλώς το μέγεθος του πίνακα, $O(mn)$.

Και στο παράδειγμά μας, η διορθωτική απόσταση προκύπτει ότι είναι 6:

```
E  X  P  O  N  E  N  —  T  I  A  L
—  —  P  O  L  Y  N  O  M  I  A  L
```

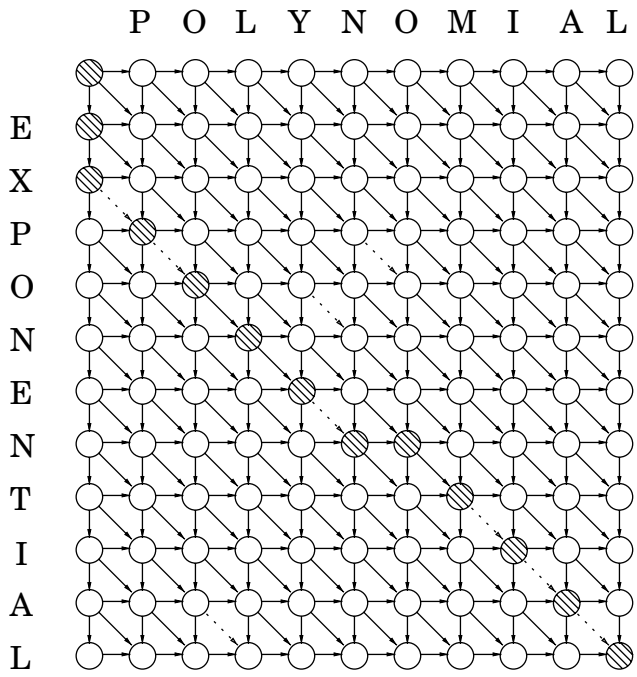
Το υποκείμενο DAG

Κάθε δυναμικό πρόγραμμα έχει μια υποκείμενη δομή DAG: θεωρήστε ότι κάθε κόμβος αναπαριστάνει ένα υποπρόβλημα και κάθε ακμή έναν περιορισμό προτεραιότητας ο οποίος αφορά τη σειρά χειρισμού των υποπροβλημάτων. Η ύπαρξη κόμβων $u_1, ..., u_k$ οι

οποίοι δείχνουν στο *ν* σημαίνει ότι «το υποπρόβλημα *ν* μπορεί να λυθεί μόνο αν είναι γνωστές οι απαντήσεις των u_1, ..., u_k».

Στη συγκεκριμένη εφαρμογή της διορθωτικής απόστασης, οι κόμβοι του υποκείμενου DAG αντιστοιχούν σε υποπροβλήματα, ή ισοδύναμα σε θέσεις (i, j) στον πίνακα. Οι ακμές είναι οι περιορισμοί προτεραιότητας, της μορφής $(i - 1, j) \rightarrow (i, j)$, $(i, j - 1) \rightarrow (i, j)$, και $(i - 1, j - 1) \rightarrow (i, j)$ (Εικόνα 6.5). Στην πραγματικότητα, μπορούμε να προχωρήσουμε τα πράγματα πιο πολύ και να τοποθετήσουμε βάρη στις ακμές έτσι ώστε οι διορθωτικές αποστάσεις να δίνονται από τις συντομότερες διαδρομές στο DAG! Για να το δείτε αυτό, θέστε σε όλες τις ακμές μήκος 1, εκτός από τις ακμές $\{(i - 1, j - 1) \rightarrow (i, j)$: $x[i] = y[j]\}$ (φαίνονται με διακεκομμένη γραμμή στην εικόνα) των οποίων το μήκος είναι 0. Τότε η τελική απάντηση είναι απλώς η απόσταση μεταξύ των κόμβων $s = (0, 0)$ και $t = (m, n)$. Παρουσιάζεται μια πιθανή συντομότερη διαδρομή, αυτή που δίνει τη στοίχιση την οποία βρήκαμε νωρίτερα. Σε αυτή τη διαδρομή κάθε κίνηση προς τα κάτω είναι διαγραφή, κάθε κίνηση προς τα δεξιά είναι εισαγωγή, και κάθε διαγώνια κίνηση είναι είτε ταίριασμα είτε αντικατάσταση.

Εικόνα 6.5 Το υποκείμενο DAG, και μια διαδρομή μήκους 6.



Τροποποιώντας τα βάρη σε αυτό το DAG μπορούμε να επιτρέψουμε γενικευμένες μορφές διορθωτικών αποστάσεων, στις οποίες οι εισαγωγές, οι διαγραφές, και οι αντικαταστάσεις θα έχουν διαφορετικά συσχετισμένα κόστη.

6.4 Σακίδιο

Κατά τη διάρκεια μιας κλοπής, ένας διαρρήκτης βρίσκει περισσότερα λάφυρα από όσα περίμενε και πρέπει να αποφασίσει τι θα πάρει. Η τσάντα του (ή το «σακίδιο» — knapsack) μπορεί να μεταφέρει συνολικό βάρος μέχρι W κιλά. Υπάρχουν n είδη από τα οποία μπορεί να επιλέξει, με βάρη $w_1,\ldots, w_n$ και χρηματική αξία $v_1,\ldots, v_n$. Ποιος είναι ο πλέον πολύτιμος συνδυασμός που μπορεί να τοποθετήσει στην τσάντα του;[1]

Για παράδειγμα, έστω $W = 10$ και

Είδος	Βάρος	Τιμή
1	6	$30
2	3	$14
3	4	$16
4	2	$9

Υπάρχουν δύο εκδοχές αυτού του προβλήματος. Αν υπάρχουν διαθέσιμες απεριόριστες ποσότητες από κάθε είδος, η βέλτιστη λύση είναι να διαλέξουμε το είδος 1 και δύο τεμάχια από το είδος 4 (σύνολο: $48). Από την άλλη πλευρά, αν υπάρχει μόνο ένα στοιχείο από κάθε είδος (για παράδειγμα, ο κλέφτης έχει διαρρήξει μια γκαλερί τέχνης), τότε το βέλτιστο σακίδιο περιέχει τα είδη 1 και 3 (σύνολο: $46).

Όπως θα δούμε στο Κεφάλαιο 8, καμία εκδοχή αυτού του προβλήματος δεν είναι πιθανό να έχει αλγόριθμο πολυωνυμικού χρόνου. Ωστόσο, με τη χρήση δυναμικού προγραμματισμού μπορούν και οι δύο να λυθούν σε χρόνο $O(nW)$, που είναι εύλογος όταν το W είναι μικρό, αλλά δεν είναι πολυωνυμικός αφού το μέγεθος της εισόδου είναι ανάλογο του $\log W$ αντί του W.

[1] Αν αυτή η εφαρμογή σάς φαίνεται κάπως απλοϊκή, αντικαταστήστε το «βάρος» με «χρόνο CPU» και τη φράση «μπορεί να μεταφέρει μόνο W κιλά» με τη φράση «υπάρχουν διαθέσιμες μόνο W χρονικές μονάδες της CPU», κ.λπ. Το πρόβλημα του σακιδίου γενικεύει μια μεγάλη ποικιλία εργασιών επιλογής πόρων με περιορισμούς.

Συνήθη υποπροβλήματα

Η εύρεση του σωστού υποπροβλήματος χρειάζεται δημιουργικότητα και πειραματισμό. Υπάρχουν όμως μερικές τυποποιημένες επιλογές που φαίνεται ότι ανακύπτουν επανειλημμένα στο δυναμικό προγραμματισμό.

i. Η είσοδος είναι $x_1, x_2, ..., x_n$ και ένα υποπρόβλημα είναι το $x_1, x_2, ..., x_i$.

$$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6} \quad x_7 \quad x_8 \quad x_9 \quad x_{10}$$

Επομένως, το πλήθος των υποπροβλημάτων είναι γραμμικό.

ii. Η είσοδος είναι $x_1, ..., x_n$ και $y_1, ..., y_m$. Ένα υποπρόβλημα είναι το $x_1, ..., x_i$ και $y_1, ..., y_j$.

$$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6} \quad x_7 \quad x_8 \quad x_9 \quad x_{10}$$
$$\boxed{y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5} \quad y_6 \quad y_7 \quad y_8$$

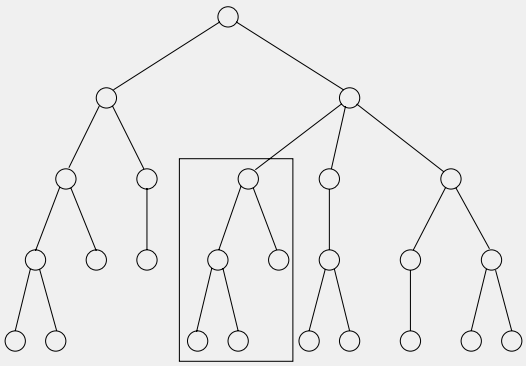
Το πλήθος των υποπροβλημάτων είναι $O(mn)$.

iii. Η είσοδος είναι $x_1, ..., x_n$ και ένα υποπρόβλημα είναι το $x_i, x_{i+1}, ..., x_j$.

$$x_1 \quad x_2 \quad \boxed{x_3 \quad x_4 \quad x_5 \quad x_6} \quad x_7 \quad x_8 \quad x_9 \quad x_{10}$$

Το πλήθος των υποπροβλημάτων είναι $O(n^2)$.

iv. Η είσοδος είναι ένα δένδρο με ρίζα. Το υποπρόβλημα είναι ένα υποδένδρο με ρίζα.



Αν το δένδρο έχει n κόμβους, πόσα υποπροβλήματα υπάρχουν;

Έχουμε ήδη συναντήσει τις δύο πρώτες περιπτώσεις, και οι άλλες θα παρουσιαστούν σύντομα.

Άνθρωποι και ποντίκια

Το σώμα μας είναι εξαιρετική μηχανή: ευέλικτο στη λειτουργία, προσαρμόσιμο σε νέα περιβάλλοντα, και σε θέση να αλληλεπιδρά και να αναπαράγεται. Όλες αυτές οι δυνατότητες καθορίζονται από ένα πρόγραμμα μοναδικό στον καθένα από εμάς, μια συμβολοσειρά μήκους 3 δισεκατομμυρίων χαρακτήρων από το αλφάβητο $\{A, C, G, T\}$ — το DNA μας.

Οι ακολουθίες DNA δύο οποιωνδήποτε ανθρώπων διαφέρουν μόνο κατά περίπου 0.1%. Ωστόσο, αυτό το ποσοστό αφήνει 3 εκατομμύρια θέσεις στις οποίες υπάρχει διαφοροποίηση, περισσότερες από αρκετές για να εξηγήσουν την τεράστια ποικιλία που εμφανίζουν οι άνθρωποι. Αυτές οι διαφορές παρουσιάζουν μεγάλο επιστημονικό και ιατρικό ενδιαφέρον — για παράδειγμα, μπορεί να μας βοηθήσουν να προβλέπουμε ποιοι άνθρωποι είναι επιρρεπείς σε ορισμένες ασθένειες.

Το DNA είναι ένα τεράστιο και φαινομενικά ανεξιχνίαστο πρόγραμμα, αλλά μπορεί να διασπαστεί σε μικρότερες μονάδες οι οποίες έχουν πιο ειδικό ρόλο, κάτι σαν υπορουτίνες. Αυτές οι μονάδες ονομάζονται *γονίδια* (genes). Οι υπολογιστές έχουν καταστεί κρίσιμο εργαλείο στην κατανόηση των γονιδίων του ανθρώπου και άλλων οργανισμών, σε τέτοιο βαθμό που σήμερα η *υπολογιστική διερεύνηση του γονιδιώματος* (computational genomics) αποτελεί ξεχωριστό επιστημονικό κλάδο. Ακολουθούν παραδείγματα τυπικών ερωτημάτων που ανακύπτουν.

1. Όταν ανακαλύπτεται ένα νέο γονίδιο, ένας τρόπος για να αποκομίσουμε πληροφορίες για τη λειτουργία του είναι να βρούμε γνωστά γονίδια τα οποία ταιριάζουν αρκετά με αυτό. Κάτι τέτοιο βοηθά ιδιαίτερα στη μεταφορά γνώσεων από είδη που έχουν μελετηθεί καλά, όπως τα ποντίκια, στον άνθρωπο.

   Μια βασική παράμετρος σε αυτό το πρόβλημα αναζήτησης είναι να ορίσουμε μια έννοια, η οποία να μπορεί να υπολογιστεί αποδοτικά, για το πότε προσεγγιστικά ταιριάζουν δύο συμβολοσειρές. Η βιολογία προτείνει μια γενίκευση της διορθωτικής απόστασης, και ο δυναμικός προγραμματισμός μπορεί να χρησιμοποιηθεί για τον υπολογισμό της.

   Κατόπιν υπάρχει το πρόβλημα της αναζήτησης μέσα από το τεράστιο σύνολο των γνωστών γονιδίων: η βάση δεδομένων GenBank έχει ήδη συνολικό μήκος μεγαλύτερο από 10^{10}, και αυτός ο αριθμός μεγαλώνει ταχύτατα. Η τρέχουσα προτιμώμενη μέθοδος είναι η BLAST, ένας ευφυής συνδυασμός αλγοριθμικών τεχνασμάτων και βιολογικών γνώσεων, με αποτέλεσμα να είναι το πλέον ευρέως χρησιμοποιούμενο λογισμικό στην υπολογιστική βιολογία.

2. Οι μέθοδοι για τον προσδιορισμό της συμβολοσειράς των χαρακτήρων που αποτελούν το DNA (*sequencing*) κατά κανόνα βρίσκουν μόνο θραύσματα των 500–700 χαρακτήρων. Μπορούν να παραχθούν δισεκατομμύρια από αυτά τα τυχαία διάσπαρτα θραύσματα, αλλά πώς μπορούν να συναρμολογηθούν σε μια ακολουθία DNA η οποία να έχει συνοχή; Καταρχήν, η θέση οποιουδήποτε θραύσματος στην τελική ακολουθία είναι άγνωστη και πρέπει να συναχθεί από τη συναρμολόγηση αλληλεπικαλυπτόμενων θραυσμάτων.

Ένα αριστούργημα αυτών των προσπαθειών είναι το προσχέδιο του ανθρώπινου DNA το οποίο ολοκληρώθηκε το 2001 από δύο ομάδες ταυτόχρονα: τη δημόσια χρηματοδοτούμενη Human Genome Consortium και την ιδιωτική Celera Genomics.

3. Όταν έχει προσδιοριστεί η συμβολοσειρά των χαρακτήρων που αποτελούν ένα συγκεκριμένο γονίδιο σε κάθε ένα από αρκετά είδη, μπορεί αυτή η πληροφορία να χρησιμοποιηθεί για να ανακατασκευαστεί η ιστορία εξέλιξης αυτού του είδους;

Θα εξερευνήσουμε αυτά τα προβλήματα στις ασκήσεις στο τέλος αυτού του κεφαλαίου. Ο δυναμικός προγραμματισμός έχει αποδειχθεί ανεκτίμητο εργαλείο για κάποια από αυτά, καθώς και για την υπολογιστική βιολογία γενικότερα.

Σακίδιο με επανάληψη

Ας ξεκινήσουμε από την εκδοχή η οποία επιτρέπει επανάληψη. Όπως πάντοτε, το κύριο ερώτημα στο δυναμικό προγραμματισμό είναι «ποια είναι τα υποπροβλήματα»; Στη συγκεκριμένη περίπτωση μπορούμε να συρρικνώσουμε το αρχικό πρόβλημα με δύο τρόπους: μπορούμε να εξετάσουμε είτε την περίπτωση σακιδίων μικρότερης χωρητικότητας $w \leq W$, ή την περίπτωση λιγότερων ειδών (για παράδειγμα, τα είδη 1, 2, ..., j, για $j \leq n$). Συνήθως χρειάζεται να πειραματιστούμε για λίγο ώστε να βρούμε ποιος ακριβώς λειτουργεί.

Ο πρώτος περιορισμός επιβάλλει μικρότερες χωρητικότητες. Συνακόλουθα, ορίζουμε

$K(w)$ = η μέγιστη τιμή η οποία επιτυγχάνεται με ένα σακίδιο χωρητικότητας w.

Μπορούμε να εκφράσουμε αυτόν τον ορισμό συναρτήσει μικρότερων υποπροβλημάτων; Αν η βέλτιστη λύση του $K(w)$ περιλαμβάνει το είδος i, τότε η αφαίρεση αυτού του είδους από το σακίδιο αφήνει μια βέλτιστη λύση για το πρόβλημα $K(w - w_i)$. Με άλλα λόγια, το $K(w)$ είναι απλώς $K(w - w_i) + v_i$, για κάποιο i. Δε γνωρίζουμε ποιο είναι αυτό το i, οπότε θα πρέπει να δοκιμάσουμε όλες τις δυνατότητες.

$$K(w) = \max_{i:w_i \leq w} \{K(w - w_i) + v_i\}$$

όπου ως συνήθως η σύμβασή μας είναι ότι η μέγιστη τιμή ενός κενού συνόλου είναι 0. Τελειώσαμε! Ο αλγόριθμος τώρα γράφεται μόνος του, και είναι ιδιαίτερα απλός και κομψός.

```
K (0) = 0
for w = 1 to W:
    K (w) = max{K (w – w_i ) + v_i: w_i ≤ w}
return K (W)
```

Αυτός ο αλγόριθμος συμπληρώνει ένα μονοδιάστατο πίνακα μήκους $W + 1$, από τα αριστερά προς τα δεξιά. Για τον υπολογισμό κάθε καταχώρισης μπορεί να χρειαστεί χρόνος το πολύ $O(n)$, οπότε ο συνολικός χρόνος εκτέλεσης είναι $O(nW)$.

Όπως πάντα, υπάρχει ένα υποκείμενο DAG. Προσπαθήστε να το κατασκευάσετε, και θα ανταμειφθείτε με μια αναπάντεχη ανακάλυψη: αυτή η συγκεκριμένη παραλλαγή του σακιδίου καταλήγει στην εύρεση της μεγαλύτερης διαδρομής σε ένα DAG!

Σακίδιο χωρίς επανάληψη

Ερχόμαστε τώρα στη δεύτερη παραλλαγή: τι συμβαίνει όταν δεν επιτρέπονται οι επαναλήψεις; Τα προηγούμενα υποπροβλήματά μας γίνονται τώρα εντελώς άχρηστα. Για παράδειγμα, η γνώση ότι η τιμή $K(w - w_n)$ είναι πολύ υψηλή δε μας βοηθά, επειδή δεν γνωρίζουμε αν το είδος n έχει ήδη χρησιμοποιηθεί σε αυτή τη μερική λύση. Επομένως, πρέπει να βελτιώσουμε την έννοια του υποπροβλήματος ώστε να μεταφέρει επιπλέον πληροφορίες σχετικά με τα είδη που χρησιμοποιούνται. Προσθέτουμε μια δεύτερη παράμετρο, $0 \le j \le n$:

$K(w)$ = μέγιστη τιμή η οποία επιτυγχάνεται με τη χρήση ενός σακιδίου χωρητικότητας w και με είδη $1,\ldots, j$.

Η απάντηση που ψάχνουμε είναι $K(W, n)$.

Πώς μπορούμε να εκφράσουμε ένα υποπρόβλημα $K(w, j)$ συναρτήσει μικρότερων υποπροβλημάτων; Αρκετά απλά: είτε το είδος j είναι απαραίτητο για να επιτύχουμε τη βέλτιστη τιμή, είτε δεν είναι απαραίτητο:

$$K(w, j) = \max\{K(w - w_j, j - 1) + v_j, K(w, j - 1)\}.$$

(Η πρώτη περίπτωση καλείται μόνο όταν $w_i \le w$.) Με άλλα λόγια, μπορούμε να εκφράσουμε το υποπρόβλημα $K(w, j)$ συναρτήσει των υποπροβλημάτων $K(\cdot, j - 1)$.

Ο αλγόριθμος λοιπόν συνίσταται στη συμπλήρωση ενός διδιάστατου πίνακα με $W + 1$ γραμμές και $n + 1$ στήλες. Κάθε καταχώριση του πίνακα χρειάζεται σταθερό χρόνο, οπότε παρόλο που ο πίνακας είναι αρκετά μεγαλύτερος από αυτόν της προηγούμενης περίπτωσης, ο χρόνος εκτέλεσης παραμένει ο ίδιος, $O(nW)$. Ακολουθεί ο κώδικας.

```
Αρχικοποίηση όλων των K(0, j) = 0 και όλων των K(w, 0) = 0
for j = 1 to n:
    for w = 1 to W:
        if w_j > w: K(w, j) = K(w, j − 1)
        else: K(w, j) = max{K(w, j − 1), K(w − w_j, j − 1) + v_j}
return K(W, n)
```

Το βιβλίο αυτό, που έχει δοκιμαστεί εκτενώς για περισσότερο από μία δεκαετία στις αίθουσες των Πανεπιστημίων της California, Berkeley και San Diego, εξηγεί τις θεμελιώδεις έννοιες των αλγορίθμων με αφηγηματικό τρόπο, με αποτέλεσμα το εκπαιδευτικό υλικό να γίνεται ευχάριστο και εύκολα κατανοητό.

Δίνεται έμφαση στην κατανόηση της κρίσιμης μαθηματικής ιδέας στην οποία βασίζεται κάθε αλγόριθμος, με τρόπο που είναι διαισθητικός και μαθηματικά ακριβής, χωρίς να είναι υπερβολικά τυπικός.

Χαρακτηριστικά του βιβλίου:

- Η χρήση πλαισίων κειμένου που ενισχύουν την αφήγηση, παρέχουν ιστορικές λεπτομέρειες και περιγραφές του τρόπου με τον οποίο χρησιμοποιούνται οι αλγόριθμοι στην πράξη, και προσφέρουν στους εξοικειωμένους με τα μαθηματικά αναγνώστες ευκαιρίες για περιηγήσεις.
- Προσεκτικά επιλεγμένα προηγμένα θέματα, τα οποία είναι δυνατό να παραλειφθούν σε ένα κανονικό εξαμηνιαίο μάθημα, αλλά μπορεί να καλυφθούν σε ένα μάθημα αλγορίθμων ανωτέρου επιπέδου ή σε μια λιγότερο εντατική κάλυψη των αλγορίθμων σε δύο εξάμηνα σπουδών.
- Μια προσιτή εξέταση του γραμμικού προγραμματισμού, η οποία εισάγει τους φοιτητές σε ένα από τα μεγαλύτερα επιτεύγματα στον τομέα των αλγορίθμων, καθώς και ένα προαιρετικό κεφάλαιο για τον κβαντικό αλγόριθμο παραγοντοποίησης, το οποίο προσφέρει μια μοναδική ευκαιρία εξερεύνησης του συναρπαστικού αυτού θέματος.

εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ
Δομοκού 4, Σταθμός Λαρίσης, 10440 ΑΘΗΝΑ, Τηλ. 210-5227035

Επισκεφθείτε μας στο Internet:
www.klidarithmos.gr

ISBN 978-960-461-211-6