Περιεχόμενα

ΜΕΡΟΣ ΔΥΟ Δομές δεδομένων 87

Πρόλογος

ΑΥΤΟ ΤΟ ΒΙΒΛΙΟ, στην πρωτότυπη αμερικανική του έκδοση, είναι το πρώτο από μια σειρά τριών τόμων που έχουν σκοπό τη μελέτη των σημαντικότερων αλγορίθμων που χρησιμοποιούνται σήμερα στους υπολογιστές. Ο πρώτος τόμος (Μέρη 1-4) καλύπτει τις θεμελιώδεις έννοιες (Μέρος 1), τις δομές δεδομένων (Μέρος 2), τους αλγορίθμους ταξινόμησης (Μέρος 3), και τους αλγορίθμους αναζήτησης (Μέρος 4). Ο δεύτερος τόμος (Μέρος 5) καλύπτει τους γράφους και τους αλγορίθμους γράφων, ενώ ο τρίτος τόμος (Μέρη 6-8) — ο οποίος δεν είχε κυκλοφορήσει ακόμη όταν μεταφραζόταν αυτό το βιβλίο — καλύπτει τα αλφαριθμητικά (Μέρος 6), την υπολογιστική γεωμετρία (Μέρος 7), και προηγμένους αλγορίθμους και εφαρμογές (Μέρος 8).

Τα συγκεκριμένα βιβλία είναι χρήσιμα ως συγγράμματα σχετικά νωρίς στο πρόγραμμα μαθημάτων της επιστήμης των υπολογιστών, αφού αποκτήσουν οι σπουδαστές βασικές προγραμματιστικές δεξιότητες και οικειότητα με τα συστήματα υπολογιστών, αλλά πριν αρχίσουν να παρακολουθούν εξειδικευμένα μαθήματα σε προηγμένους τομείς της επιστήμης των υπολογιστών ή των εφαρμογών υπολογιστών. Τα βιβλία είναι επίσης χρήσιμα και ως μέσα αυτοδιδασκαλίας ή ως οδηγοί αναφοράς για τα άτομα που ασχολούνται με την ανάπτυξη συστημάτων υπολογιστών ή προγραμμάτων εφαρμογών, επειδή περιέχουν υλοποιήσεις χρήσιμων αλγορίθμων και λεπτομερείς πληροφορίες για τα χαρακτηριστικά επιδόσεων αυτών των αλγορίθμων. Η ευρεία προοπτική που ακολουθείται κάνει αυτή τη σειρά βιβλίων κατάλληλη ως εισαγωγή στο συγκεκριμένο γνωστικό πεδίο.

Στο σύνολό τους, οι τρεις τόμοι αποτελούν την *Τρίτη Αμερικανική Έκδοση* ενός βιβλίου το οποίο χρησιμοποιείται ευρέως από σπουδαστές και προγραμματιστές σε ολόκληρο τον κόσμο για πολλά χρόνια. Έχω ξαναγράψει από την αρχή το κείμενο αυτής της έκδοσης και έχω προσθέσει χιλιάδες νέες ασκήσεις, εκατοντάδες νέες εικόνες, δεκάδες νέα προγράμματα, και λεπτομερή σχόλια σε όλες τις εικόνες και τα προγράμματα. Στόχος μου με την προσθήκη αυτού του νέου υλικού ήταν να καλύψω νέα θέματα αλλά και να εξηγήσω αναλυτικότερα πολλούς από τους κλασικούς αλγορίθμους. Η έμφαση που δίνεται (σε όλα τα βιβλία) στους αφηρημένους τύπους δεδομένων, κάνει τα προγράμματα χρήσιμα σε ένα μεγαλύτερο εύρος εφαρμογών και κατάλληλα για σύγχρονα αντικειμενοστρεφή περιβάλλοντα προγραμματισμού. Οι αναγνώστες που έχουν διαβάσει τις προηγούμενες εκδόσεις θα βρουν σε αυτή τη νέα έκδοση αρκετές νέες πληροφορίες, και όλοι οι αναγνώστες θα βρουν πλούσιο διδακτικό υλικό που παρέχει αποτελεσματική πρόσβαση στις βασικές έννοιες.

Αυτά τα βιβλία δεν είναι κατάλληλα μόνο για προγραμματιστές και σπουδαστές της επιστήμης των υπολογιστών. Όλοι όσοι χρησιμοποιούν υπολογιστές θέλουν τα προγράμματα που χρησιμοποιούν να εκτελούνται ταχύτερα ή να λύνουν μεγαλύτερα προβλήματα. Οι αλγόριθμοι που παρουσιάζουμε αντιπροσωπεύουν γνώσεις οι οποίες αναπτύχθηκαν τα τελευταία

50 χρόνια και αποτελούν τη βάση για αποδοτική χρήση του υπολογιστή σε μια ευρεία κλίμακα εφαρμογών. Από τα προβλήματα προσομοίωσης του προβλήματος των N σωμάτων στη φυσική μέχρι τα προβλήματα των γενετικών ακολουθιών στη μοριακή βιολογία, οι βασικές μέθοδοι που περιγράφονται εδώ έχουν γίνει απαραίτητες στην επιστημονική έρευνα· επίσης, από τα συστήματα βάσεων δεδομένων μέχρι της μηχανές αναζήτησης στο Διαδίκτυο, έχουν γίνει απαραίτητα τμήματα των σύγχρονων συστημάτων λογισμικού. Καθώς το πεδίο των εφαρμογών υπολογιστών γίνεται όλο και μεγαλύτερο, ταυτόχρονα αυξάνεται και η επίδραση των βασικών αλγορίθμων. Στόχος του παρόντος βιβλίου είναι να λειτουργήσει ως πηγή αναφοράς, έτσι ώστε οι σπουδαστές και οι επαγγελματίες να γνωρίσουν αυτούς τους θεμελιώδεις αλγορίθμους προκειμένου να μπορούν να τους χρησιμοποιούν με έξυπνους τρόπους όποτε παρουσιάζεται ανάγκη, σε οποιαδήποτε εφαρμογή υπολογιστή.

Στόχοι του βιβλίου

Το βιβλίο *Αλγόριθμοι σε C*, *Τρίτη αμερικανική έκδοση*, *Μέρη 1-4*, αποτελείται από 16 κεφάλαια τα οποία έχουν ομαδοποιηθεί σε τέσσερα κύρια μέρη: θεμελιώδεις έννοιες, δομές δεδομένων, ταξινόμηση, και αναζήτηση. Το βιβλίο έχει στόχο να δώσει στους αναγνώστες τη δυνατότητα να κατανοήσουν τις βασικές ιδιότητες όσο το δυνατό περισσότερων θεμελιωδών αλγορίθμων. Οι αλγόριθμοι που περιγράφονται χρησιμοποιούνται ευρέως για πολλά χρόνια, και αντιπροσωπεύουν ένα βασικό τμήμα γνώσης τόσο για τον ασκούμενο προγραμματιστή όσο και για το σπουδαστή της επιστήμης των υπολογιστών. Ο δεύτερος τόμος είναι αφιερωμένος σε αλγορίθμους γράφων, και ο τρίτος αποτελείται από τέσσερα μέρη που καλύπτουν τα αλφαριθμητικά, τη γεωμετρία, και προχωρημένα θέματα. Ο κύριος στόχος μου στη συγγραφή αυτών των βιβλίων ήταν να συγκεντρώσω τις θεμελιώδεις μεθόδους αυτών των τομέων της επιστήμης των υπολογιστών, έτσι ώστε να κάνω προσιτές τις καλύτερες γνωστές μεθόδους για την επίλυση προβλημάτων με υπολογιστή.

Θα εκτιμήσετε ακόμη περισσότερο το υλικό του βιβλίου αν έχετε ήδη παρακολουθήσει ένα ή δύο προηγούμενα μαθήματα της επιστήμης των υπολογιστών ή διαθέτετε αντίστοιχη εμπειρία στον προγραμματισμό: ένα μάθημα στον προγραμματισμό σε γλώσσα υψηλού επιπέδου όπως η C, η C++, ή η Java, και ίσως ένα ακόμη μάθημα στο οποίο διδάσκονται οι θεμελιώδεις έννοιες των συστημάτων προγραμματισμού. Κατά συνέπεια, το βιβλίο απευθύνεται σε όλους εκείνους που έχουν επαφή με κάποια σύγχρονη γλώσσα προγραμματισμού και σε εκείνους που ασχολούνται με τα βασικά χαρακτηριστικά των σύγχρονων συστημάτων υπολογιστών. Επίσης, στο κείμενο θα βρείτε παραπομπές σε βιβλιογραφία που ίσως σας βοηθήσει να καλύψετε κάποια κενά στα θέματα που περιγράφονται.

Επειδή το μεγαλύτερο μέρος της ύλης των μαθηματικών με το οποίο υποστηρίζονται τα αναλυτικά αποτελέσματα είναι αυτοτελές (ή επισημαίνεται ότι ξεφεύγει από τους στόχους του βιβλίου), δεν χρειάζεται πολλή προετοιμασία στα μαθηματικά για το μεγαλύτερο τμήμα του βιβλίου — αν και η μαθηματική ωριμότητα οπωσδήποτε είναι χρήσιμη.

Χρήση του βιβλίου ως διδακτέας ύλης

Υπάρχει μεγάλη ευελιξία σε ό,τι αφορά τον τρόπο με τον οποίο μπορεί να διδαχθεί η ύλη του βιβλίου· εξαρτάται από τον καθηγητή και την προετοιμασία των σπουδαστών του. Γίνεται επαρκής κάλυψη της βασικής ύλης, έτσι ώστε να μπορεί το βιβλίο να χρησιμοποιηθεί για τη διδασκαλία αρχαρίων στις δομές δεδομένων, ενώ ταυτόχρονα παρέχονται επαρκείς λεπτομέρειες και κάλυψη του προχωρημένου υλικού προκειμένου να μπορεί να χρησιμοποιηθεί και στη διδασκαλία της σχεδίασης και ανάλυσης αλγορίθμων σε σπουδαστές πιο προχωρημένου επιπέδου. Μερικοί καθηγητές μπορεί να προτιμήσουν να δώσουν έμφαση στις υλοποιήσεις και στα πρακτικά θέματα, ενώ κάποιοι άλλοι μπορεί να επιθυμούν να δώσουν έμφαση στην ανάλυση και τις θεωρητικές έννοιες.

Σε ένα εισαγωγικό μάθημα στις δομές δεδομένων και τους αλγορίθμους, θα μπορούσε να δοθεί έμφαση στις βασικές δομές δεδομένων του Μέρους 2 και τη χρήση τους στις υλοποιήσεις των Μερών 3 και 4. Σε ένα μάθημα σχεδίασης και ανάλυσης αλγορίθμων θα μπορούσε να δοθεί έμφαση στο θεμελιώδες υλικό του Μέρους 1 και στο Κεφάλαιο 5, και στη συνέχεια να μελετηθούν οι τρόποι με τους οποίους οι αλγόριθμοι των Μερών 3 και 4 επιτυγχάνουν καλές ασυμπτωτικές επιδόσεις. Σε ένα μάθημα τεχνολογίας λογισμικού, θα μπορούσε να παραλειφθεί η ύλη των μαθηματικών καθώς και η προχωρημένη ύλη των αλγορίθμων, και να δοθεί έμφαση στους τρόπους ολοκλήρωσης των υλοποιήσεων που περιλαμβάνονται στο βιβλίο σε μεγάλα προγράμματα ή συστήματα. Τέλος, σε ένα μάθημα με θέμα τους αλγορίθμους, θα μπορούσε να ακολουθηθεί μια γενική προσέγγιση και να εισαχθούν έννοιες από όλους τους παραπάνω τομείς.

Οι προηγούμενες εκδόσεις του βιβλίου, οι οποίες βασίζονται σε άλλες γλώσσες προγραμματισμού, έχουν χρησιμοποιηθεί σε πάρα πολλά κολέγια και πανεπιστήμια ως διδακτέα ύλη για το δεύτερο ή τρίτο μάθημα στην επιστήμη των υπολογιστών, και ως βοηθητικό εγχειρίδιο για άλλα μαθήματα. Στο Princeton, έχουμε διαπιστώσει ότι η ύλη που καλύπτει το βιβλίο παρέχει στους σπουδαστές μας μια εισαγωγή στην επιστήμη των υπολογιστών η οποία μπορεί να επεκταθεί σε μεταγενέστερα μαθήματα που αφορούν την ανάλυση αλγορίθμων, τον προγραμματισμό συστημάτων, και τη θεωρητική επιστήμη των υπολογιστών· από την άλλη μεριά, παρέχει στην αυξανόμενη ομάδα των σπουδαστών από άλλους επιστημονικούς κλάδους ένα ευρύ σύνολο τεχνικών τις οποίες μπορούν να χρησιμοποιήσουν αμέσως.

Οι ασκήσεις — από τις οποίες σχεδόν όλες είναι καινούργιες σε αυτή την τρίτη έκδοση — ανήκουν σε διάφορους τύπους. Μερικές έχουν στόχο να ελέγξουν αν έγινε κατανοητή η ύλη του βιβλίου, και απλώς ζητούν από τους αναγνώστες να εργαστούν σε κάποιο παράδειγμα ή να εφαρμόσουν βασικές έννοιες που περιγράφονται στο βιβλίο. Άλλες περιλαμβάνουν την υλοποίηση και τη συναρμολόγηση των αλγορίθμων, ή την εκτέλεση πειραματικών μελετών με στόχο τη σύγκριση διαφόρων παραλλαγών των αλγορίθμων και την καλύτερη κατανόηση των ιδιοτήτων τους. Κάποιες άλλες, πάλι, είναι ένας θησαυρός σημαντικών πληροφοριών, σε επίπεδο λεπτομέρειας το οποίο δεν είναι κατάλληλο για το βιβλίο. Η μελέτη και η διερεύνηση των ασκήσεων θα βοηθήσουν σε μεγάλο βαθμό τον αναγνώστη να κατανοήσει τα θέματα που περιγράφονται στο βιβλίο.

Αλγόριθμοι πρακτικής χρήσης

Οποιοσδήποτε θέλει να χρησιμοποιήσει τον υπολογιστή του αποτελεσματικότερα μπορεί να χρησιμοποιήσει αυτό το βιβλίο ως οδηγό αναφοράς ή για αυτοδιδασκαλία. Όσοι έχουν πείρα στον προγραμματισμό μπορούν να βρουν στα κεφάλαια του βιβλίου πληροφορίες που αφορούν συγκεκριμένα θέματα. Ως ένα μεγάλο βαθμό, μπορείτε να διαβάσετε κάθε κεφάλαιο του βιβλίου ανεξάρτητα από τα υπόλοιπα, παρά το γεγονός ότι, σε μερικές περιπτώσεις, ορισμένοι αλγόριθμοι ενός κεφαλαίου χρησιμοποιούν μεθόδους από κάποιο προηγούμενο κεφάλαιο.

Το βιβλίο είναι προσανατολισμένο στη μελέτη των αλγορίθμων που είναι πιθανό να έχουν πρακτική χρήση. Παρέχονται πληροφορίες σχετικά με τα εργαλεία του εμπορίου, έτσι ώστε οι αναγνώστες να μπορούν με βεβαιότητα να υλοποιήσουν, να αποσφαλματώσουν, και να χρησιμοποιήσουν στην πράξη αλγορίθμους για την επίλυση ενός προβλήματος ή την προσθήκη επιπλέον λειτουργιών σε κάποια εφαρμογή. Περιλαμβάνονται πλήρεις υλοποιήσεις των μεθόδων που εξετάζονται, όπως επίσης και περιγραφές των λειτουργιών αυτών των προγραμμάτων σε ένα συνεπές σύνολο παραδειγμάτων.

Επειδή εργαζόμαστε με πραγματικό κώδικα αντί να γράφουμε ψευδοκώδικα, μπορείτε να χρησιμοποιήσετε αυτά τα προγράμματα στην πράξη πολύ γρήγορα. Τα προγράμματα είναι διαθέσιμα και στην ηλεκτρονική σελίδα του βιβλίου στο Διαδίκτυο. Μπορείτε να χρησιμοποιήσετε αυτά τα προγράμματα με πολλούς τρόπους προκειμένου να σας βοηθήσουν στη μελέτη των αλγορίθμων. Διαβάστε τα για να ελέγξετε πόσο έχετε κατανοήσει τις λεπτομέρειες ενός αλγορίθμου, ή για να δείτε κάποιον τρόπο χειρισμού ανάθεσης αρχικών τιμών, οριακών συνθηκών, και άλλων περίεργων καταστάσεων που αποτελούν συχνά προγραμματιστικές προκλήσεις. Εκτελέστε τα για να δείτε τους αλγορίθμους σε δράση, να μελετήσετε πειραματικά τις επιδόσεις τους, και να ελέγξετε τα αποτελέσματά σας σε σχέση με τους πίνακες που περιέχονται στο βιβλίο, ή να δοκιμάσετε τις δικές σας τροποποιήσεις.

Μια από τις πρακτικές εφαρμογές των αλγορίθμων είναι και η χρήση τους για την παραγωγή των εκατοντάδων εικόνων που περιλαμβάνονται σε αυτό το βιβλίο. Πολλοί αλγόριθμοι αποκαλύπτονται σε διαισθητικό επίπεδο μέσω της οπτικοποιημένης τους διάστασης η οποία παρέχεται από αυτές τις εικόνες.

Στο βιβλίο εξετάζονται λεπτομερώς τα χαρακτηριστικά των αλγορίθμων και των περιπτώσεων στις οποίες μπορεί να είναι χρήσιμοι, ενώ παράλληλα γίνονται συνδέσεις με την ανάλυση αλγορίθμων και τη θεωρητική επιστήμη των υπολογιστών. Όποτε κρίνεται απαραίτητο, παρουσιάζονται πειραματικά και αναλυτικά αποτελέσματα προκειμένου να διασαφηνιστεί ο λόγος για τον οποίο προτιμούνται κάποιοι συγκεκριμένοι αλγόριθμοι. Όποτε παρουσιάζει ενδιαφέρον, περιγράφεται και η σχέση των πρακτικών αλγορίθμων που εξετάζονται με αμιγώς θεωρητικά αποτελέσματα. Επίσης, οι ειδικές πληροφορίες σχετικά με τα χαρακτηριστικά επιδόσεων των αλγορίθμων και των υλοποιήσεων συγκεντρώνονται, ενσωματώνονται στο κείμενο, και αναλύονται σε ολόκληρο το βιβλίο.

Γλώσσα προγραμματισμού

Η γλώσσα προγραμματισμού που χρησιμοποιείται για όλες τις υλοποιήσεις των αλγορίθμων είναι η C. Κάθε γλώσσα έχει τα πλεονεκτήματα και τα μειονεκτήματά της· χρησιμοποιούμε τη C επειδή είναι ευρέως διαθέσιμη και επειδή παρέχει τις δυνατότητες που είναι απαραίτητες για τις υλοποιήσεις μας. Πάντως, τα προγράμματα είναι δυνατό να μεταφραστούν πολύ εύκολα και σε άλλες σύγχρονες γλώσσες προγραμματισμού, μιας και ο κώδικας που αφορά αποκλειστικά τη C είναι σχετικά λίγος. Όπου κρίνεται κατάλληλο, χρησιμοποιούμε και καθιερωμένους ιδιωματισμούς της C, αλλά το βιβλίο δεν έχει στόχο να αποτελέσει οδηγό αναφοράς στον προγραμματισμό σε C.

Σε αυτή τη νέα έκδοση του βιβλίου έχουν προστεθεί πολλά νέα προγράμματα, ενώ σε πολλά από τα παλιότερα έχουν γίνει τροποποιήσεις, κυρίως προκειμένου να γίνουν αμέσως χρήσιμα ως υλοποιήσεις αφηρημένων τύπων δεδομένων. Επίσης, σε διάφορα σημεία του κειμένου παρατίθενται και αναλύονται τα αποτελέσματα εκτεταμένων συγκριτικών πειραματικών δοκιμών των προγραμμάτων.

Ένας από τους στόχους αυτού του βιβλίου είναι η παρουσίαση των αλγορίθμων σε όσο το δυνατό απλούστερη και αμεσότερη μορφή. Το στυλ προγραμματισμού παρουσιάζει συνέπεια όποτε αυτό είναι δυνατό, έτσι ώστε τα προγράμματα που είναι παρόμοια να φαίνονται και παρόμοια. Στην περίπτωση πολλών από τους αλγορίθμους του βιβλίου, οι ομοιότητες ισχύουν ανεξάρτητα από τη γλώσσα: ο αλγόριθμος ταξινόμησης quicksort είναι πάντοτε ο αλγόριθμος ταξινόμησης quicksort (για να επιλέξουμε ένα εξέχον παράδειγμα), ανεξάρτητα από το αν εκφράζεται σε Ada, Algol-60, Basic, C, C++, Fortran, Java, Mesa, Modula-3, Pascal, PostScript, Smalltalk, ή σε οποιαδήποτε από τις αμέτρητες άλλες γλώσσες προγραμματισμού και περιβάλλοντα όπου έχει αποδειχθεί ότι είναι μια αποτελεσματική μέθοδος ταξινόμησης. Από τη μια πλευρά, ο κώδικάς μας παίρνει πληροφορίες από τις εμπειρίες υλοποίησης αλγορίθμων σε αυτές και σε πολλές άλλες γλώσσες προγραμματισμού (υπάρχουν εκδόσεις του βιβλίου και για τις γλώσσες C++ και Java)· από την άλλη, ορισμένες από τις ιδιότητες μερικών από αυτές τις γλώσσες εμπλουτίζονται από την εμπειρία των σχεδιαστών τους σε μερικούς από τους αλγορίθμους και τις δομές δεδομένων που εξετάζουμε σε αυτό το βιβλίο.

Στόχος μας σε αυτό το βιβλίο ήταν να παρουσιάσουμε έξυπνες, συμπαγείς, και φορητές υλοποιήσεις των αλγορίθμων· παράλληλα, όμως, επειδή άποψή μας είναι ότι η αποδοτικότητα έχει μεγάλη σημασία, είχαμε κατά νου τα χαρακτηριστικά επιδόσεων του κώδικά μας σε όλα τα στάδια της ανάπτυξής του. Το Κεφάλαιο 1 αποτελεί λεπτομερές παράδειγμα αυτής της προσέγγισης στην ανάπτυξη αποδοτικών υλοποιήσεων των αλγορίθμων μας σε C, ενώ στο Κεφάλαιο 2 περιγράφεται η προσέγγιση που ακολουθούμε για την ανάλυσή τους. Τα Κεφάλαια 3 και 4 είναι αφιερωμένα στην περιγραφή και την αιτιολόγηση των βασικών μηχανισμών που χρησιμοποιούμε στις υλοποιήσεις τύπων δεδομένων και αφηρημένων τύπων δεδομένων (ΑΤΔ). Αυτά τα τέσσερα κεφάλαια αποτελούν τα θεμέλια για τα επόμενα κεφάλαια του βιβλίου.

Ευχαριστίες

Η ανταπόκριση πολλών ανθρώπων από τις παλιότερες εκδόσεις αυτού του βιβλίου ήταν ιδιαίτερα χρήσιμη. Πιο συγκεκριμένα, εκατοντάδες φοιτητές των πανεπιστημίων Princeton και Brown υπέστησαν τις προκαταρκτικές εκδόσεις για αρκετά χρόνια. Θα ήθελα να εκφράσω ιδιαίτερες ευχαριστίες στην Trina Avery και τον Tom Freeman για τη βοήθειά τους στη δημιουργία της πρώτης έκδοσης· στην Janet Incerpi για τη δημιουργικότητα και την εξυπνάδα της που της επέτρεψαν να πειθαναγκάσει το πρωτόγονο υλικό και λογισμικό συγγραφής κειμένου που χρησιμοποιούσαμε προκειμένου να δημιουργήσουν την πρώτη έκδοση· στον Marc Brown για τη συμμετοχή του στην έρευνα οπτικοποίησης των αλγορίθμων, η οποία οδήγησε στη δημιουργία πολλών από τις εικόνες του βιβλίου· και στον Dave Hanson για την προθυμία που έδειξε να απαντήσει σε όλες τις ερωτήσεις μου σχετικά με τη C. Θα ήθελα επίσης να ευχαριστήσω όλους τους αναγνώστες που έκαναν σχόλια σχετικά με τις διάφορες εκδόσεις του βιβλίου, μεταξύ των οποίων είναι και οι Guy Almes, Jon Bentley, Marc Brown, Jay Gischer, Allan Heydon, Kennedy Lemke, Udi Manber, Dana Richards, John Reif, M. Rosenfeld, Stephen Seidman, Michael Quinn, και William Ward.

Για να δημιουργήσω αυτή τη νέα έκδοση είχα την ευχαρίστηση να συνεργαστώ με τους Peter Gordon και Debbie Lafferty της Addison-Wesley, οι οποίοι και καθοδήγησαν με υπομονή αυτό το έργο καθώς εξελισσόταν. Επίσης, είχα την ευχαρίστηση να συνεργαστώ με πολλά άλλα μέλη του εξειδικευμένου προσωπικού της Addison-Wesley. Η φύση του έργου έκανε το βιβλίο μια, κατά κάποιον τρόπο, ασυνήθιστη πρόκληση για πολλούς από αυτούς, και εκτιμώ ιδιαίτερα την υπομονή τους.

Κέρδισα και δύο νέους μέντορες κατά τη συγγραφή αυτού του βιβλίου, και θα ήθελα να εκφράσω ιδιαίτερα την εκτίμησή μου σε αυτούς. Ο πρώτος ήταν ο Steve Summit, ο οποίος έλεγξε προσεκτικά τις πρώτες εκδόσεις του βιβλίου σε τεχνικό επίπεδο και έκανε κυριολεκτικά χιλιάδες λεπτομερή σχόλια, ιδιαίτερα για τα προγράμματα. Ο Steve κατάλαβε ξεκάθαρα το στόχο μου να δώσω κομψές, αποδοτικές, και αποτελεσματικές υλοποιήσεις, και τα σχόλιά του, όχι μόνο με βοήθησαν να προσφέρω ένα μέτρο συνέπειας κατά τις υλοποιήσεις, αλλά και να βελτιώσω σημαντικά πολλές από αυτές. Ο δεύτερος ήταν η Lyn Dupré που έκανε και αυτή χιλιάδες λεπτομερή σχόλια, τα οποία ήταν ανεκτίμητης αξίας και με βοήθησαν, όχι μόνο να διορθώσω και να αποφύγω γραμματικά λάθη, αλλά επίσης — που είναι ακόμη σημαντικότερο — να βρω ένα περιεκτικό και συνεπές στυλ συγγραφής το οποίο με βοήθησε να συναρμολογήσω το μεγάλο όγκο του τεχνικού υλικού. Είμαι εξαιρετικά ευγνώμων για την ευκαιρία που μου δόθηκε να μάθω πολλά πράγματα από τους Steve και Lyn — η προσφορά τους ήταν ζωτικής σημασίας για την ανάπτυξη αυτού του βιβλίου.

Πολλά από αυτά που γράφονται στο βιβλίο τα έμαθα από τη διδασκαλία και τα γραπτά του Don Knuth, που ήταν ο επιβλέπων καθηγητής μου στο Πανεπιστήμιο Stanford. Παρά το γεγονός ότι ο Don δεν είχε άμεση επίδραση σε αυτή τη δουλειά, η παρουσία του γίνεται αισθητή σε αυτό το βιβλίο επειδή ήταν εκείνος που έθεσε τη μελέτη των αλγορίθμων στην επιστημονική βάση η οποία καθιστά δυνατή μια δουλειά σαν αυτή. Ο φίλος και συνάδελφός μου Philippe Flajolet, ο οποίος αποτέλεσε βασικό παράγοντα για την ανάπτυξη της ανάλυσης αλγορίθμων ως ενός ώριμου τομέα ερευνών, είχε και αυτός παρόμοια επίδραση σε αυτό το έργο.

Θα ήθελα να εκφράσω τις βαθιές μου ευχαριστίες για την υποστήριξη που είχα από το Πανεπιστήμιο Princeton, το Πανεπιστήμιο Brown, και το Εθνικό Ινστιτούτο Έρευνας στην Πληροφορική και τον Αυτοματισμό (Institut National de Recherche en Informatique et Automatique, INRIA), όπου έκανα το μεγαλύτερο μέρος της έρευνας για το βιβλίο· το ίδιο ισχύει και για το Ινστιτούτο Αναλύσεων Άμυνας (Institute for Defense Analyses) καθώς και για το Ερευνητικό Κέντρο της Xerox στο Palo Alto (Xerox Palo Alto Research Center), όπου έκανα ένα μέρος της δουλειάς όσο ήμουν επισκέπτης. Πολλά μέρη του βιβλίου βασίζονται σε έρευνες που υποστηρίχθηκαν γενναιόδωρα από το Εθνικό Ίδρυμα Ερευνών (National Science Foundation) και το Γραφείο Ναυτικών Ερευνών (Office of Naval Research). Τέλος, ευχαριστώ τους Bill Bowen, Aaron Lemonick, και Neil Rudenstine για την υποστήριξή τους στη δημιουργία ακαδημαϊκού περιβάλλοντος στο Princeton, στο οποίο είχα τη δυνατότητα να ετοιμάσω αυτό το βιβλίο παρά τις πολλές άλλες υποχρεώσεις μου.

Robert Sedgewick
Marly-le-Roi, Γαλλία, Φεβρουάριος 1983
Princeton, New Jersey, Ιανουάριος 1990
Jamestown, Rhode Island, Αύγουστος 1997

ΚΕΦΑΛΑΙΟ ΕΝΝΕΑ

Ουρές προτεραιότητας και ο αλγόριθμος heapsort

ΠΟΛΛΕΣ ΕΦΑΡΜΟΓΕΣ ΑΠΑΙΤΟΥΝ από εμάς να επεξεργαζόμαστε τις εγγραφές που έχουν κλειδιά με τη σειρά, αλλά όχι απαραίτητα πλήρως ταξινομημένες ούτε απαραίτητα όλες ταυτόχρονα. Συχνά, συγκεντρώνουμε ένα σύνολο εγγραφών και στη συνέχεια επεξεργαζόμαστε την εγγραφή με το μεγαλύτερο κλειδί, μετά ενδεχομένως συλλέγουμε περισσότερες εγγραφές και επεξεργαζόμαστε αυτή με το μεγαλύτερο τη δεδομένη στιγμή κλειδί, και ούτω καθεξής. Μια κατάλληλη δομή δεδομένων για ένα τέτοιο περιβάλλον υποστηρίζει τις λειτουργίες εισαγωγής ενός στοιχείου και διαγραφής του μεγαλύτερου στοιχείου. Μια τέτοια δομή δεδομένων ονομάζεται *ουρά προτεραιότητας* (priority queue). Η χρήση μιας ουράς προτεραιότητας μοιάζει με τη χρήση μιας ουράς (διαγραφή του παλαιότερου στοιχείου) και μιας στοίβας (διαγραφή του πιο πρόσφατου στοιχείου), αλλά η αποδοτική υλοποίησή της είναι δυσκολότερη. Η ουρά προτεραιότητας αποτελεί το σημαντικότερο παράδειγμα του γενικευμένου αφηρημένου τύπου δεδομένων (ΑΤΔ) ουράς που γνωρίσαμε στην Ενότητα 4.6. Για την ακρίβεια, η ουρά προτεραιότητας είναι μια κατάλληλη γενίκευση των δομών στοίβας και ουράς, επειδή μπορούμε να υλοποιήσουμε αυτές τις δομές δεδομένων με ουρές προτεραιότητας χρησιμοποιώντας κατάλληλες αναθέσεις προτεραιότητας στα στοιχεία (δείτε τις Ασκήσεις 9.3 και 9.4).

Ορισμός 9.1 **Ουρά προτεραιότητας** *(priority queue) είναι μια δομή δεδομένων στοιχείων με κλειδιά η οποία υποστηρίζει δύο βασικές λειτουργίες: εισαγωγή ενός νέου στοιχείου και διαγραφή του στοιχείου με το μεγαλύτερο κλειδί.*

Εφαρμογές των ουρών προτεραιότητας μπορούμε να συναντήσουμε σε συστήματα προσομοίωσης (όπου τα κλειδιά μπορεί να αντιστοιχούν σε χρόνους συμβάντων των οποίων η επεξεργασία πρέπει να γίνει με χρονολογική σειρά), στο χρονοπρογραμματισμό εργασιών σε συστήματα υπολογιστών (όπου τα κλειδιά μπορεί να αντιστοιχούν σε προτεραιότητες που δείχνουν ποιοι χρήστες πρέπει να εξυπηρετηθούν πρώτοι), και σε αριθμητικούς υπολογισμούς

(όπου τα κλειδιά μπορεί να αντιστοιχούν σε υπολογιστικά σφάλματα, δείχνοντας ότι πρέπει να αντιμετωπιστεί πρώτα το μεγαλύτερο σφάλμα).

Μπορούμε να χρησιμοποιήσουμε ουρές προτεραιότητας ως βάση κάποιου αλγορίθμου ταξινόμησης, προσθέτοντας όλες τις εγγραφές και στη συνέχεια αφαιρώντας αναδρομικά το μεγαλύτερο στοιχείο προκειμένου να ταξινομήσουμε τις εγγραφές σε αντίστροφη σειρά. Σε επόμενες ενότητες του βιβλίου θα δούμε τον τρόπο με τον οποίο μπορούμε να χρησιμοποιήσουμε ουρές προτεραιότητας ως δομικά στοιχεία για πιο προηγμένους αλγορίθμους. Στο Μέρος 5 θα δούμε γιατί οι ουρές προτεραιότητας αποτελούν μια κατάλληλη αφαίρεση η οποία μας βοηθά να κατανοήσουμε τις σχέσεις που υπάρχουν ανάμεσα σε πολλούς θεμελιώδεις αλγορίθμους αναζήτησης σε γράφους, και στο Μέρος 6 θα αναπτύξουμε έναν αλγόριθμο συμπίεσης αρχείων χρησιμοποιώντας ρουτίνες από αυτό το κεφάλαιο. Όλα αυτά δεν είναι τίποτε άλλο από μερικά παραδείγματα του σημαντικού ρόλου που παίζουν οι ουρές προτεραιότητας ως βασικό εργαλείο κατά τη σχεδίαση αλγορίθμων.

Στην πράξη, οι ουρές προτεραιότητας είναι πιο πολύπλοκες από τον απλό ορισμό που μόλις δώσαμε, επειδή υπάρχουν και μερικές άλλες λειτουργίες τις οποίες είναι πιθανό να χρειαστεί να εκτελέσουμε προκειμένου να μπορούμε να τηρούμε ουρές προτεραιότητας κάτω από οποιεσδήποτε συνθήκες οι οποίες θα μπορούσαν να εμφανιστούν κατά τη χρήση τους. Πράγματι, ένας από τους βασικούς λόγους για τον οποίο είναι τόσο χρήσιμες πολλές υλοποιήσεις ουρών προτεραιότητας είναι η ευελιξία που παρέχουν στις εφαρμογές προγραμμάτων-πελατών να εκτελούν μια ποικιλία διαφορετικών λειτουργιών σε σύνολα εγγραφών με κλειδιά. Γι' αυτόν το σκοπό, πρέπει να δημιουργήσουμε και να συντηρούμε μια δομή δεδομένων η οποία να περιέχει εγγραφές με αριθμητικά κλειδιά (*προτεραιότητες*) και να υποστηρίζει μερικές από τις επόμενες λειτουργίες:

- *Κατασκευή* μιας ουράς προτεραιότητας από N δεδομένα στοιχεία.
- *Εισαγωγή* ενός νέου στοιχείου.
- *Διαγραφή του μέγιστου* στοιχείου.
- *Αλλαγή της προτεραιότητας* ενός αυθαίρετα καθοριζόμενου στοιχείου.
- *Διαγραφή* ενός αυθαίρετα καθοριζόμενου στοιχείου.
- *Ένωση* δύο ουρών προτεραιότητας σε μια μεγαλύτερη.

Αν οι εγγραφές μπορούν να έχουν διπλά κλειδιά, θεωρούμε ότι η έννοια "μέγιστο" σημαίνει "οποιαδήποτε εγγραφή με τη μεγαλύτερη τιμή κλειδιού". Όπως και σε πολλές άλλες δομές δεδομένων, στο παραπάνω σύνολο λειτουργιών χρειάζεται επίσης να προσθέσουμε καθιερωμένες λειτουργίες *ανάθεσης αρχικών τιμών*, *ελέγχου αν είναι κενή*, και ίσως και λειτουργίες *καταστροφής* και *αντιγραφής*.

Υπάρχει κάποια αλληλοκάλυψη σε αυτές τις λειτουργίες και, μερικές φορές, είναι βολικό να ορίσουμε άλλες, παρόμοιες λειτουργίες. Για παράδειγμα, κάποιοι συγκεκριμένοι πελάτες μπορεί να χρειάζεται συχνά να *βρίσκουν το μέγιστο* στοιχείο μέσα στην ουρά προτεραιότητας χωρίς να πρέπει απαραίτητα να το αφαιρέσουν, ή μπορεί να έχουμε κάποια λειτουργία *αντικατάστασης του μέγιστου* στοιχείου με κάποιο νέο στοιχείο. Θα μπορούσαμε να υλοποιήσουμε τέτοιου είδους λειτουργίες χρησιμοποιώντας τις δύο βασικές λειτουργίες ως δομικά στοιχεία: η *εύρεση του μέγιστου* θα μπορούσε να είναι *διαγραφή του μέγιστου* ακολουθούμενη από *εισαγωγή*, και η *αντικατάσταση του μέγιστου* θα μπορούσε να είναι είτε *εισαγωγή* ακολου-

θούμενη από *διαγραφή του μέγιστου* είτε *διαγραφή του μέγιστου* ακολουθούμενη από *εισαγωγή*. Συνήθως, όμως, υλοποιώντας άμεσα τέτοιου είδους λειτουργίες καταλήγουμε σε αποδοτικότερο κώδικα, με την προϋπόθεση ότι είναι απαραίτητες και καθορίζονται ακριβώς. Ο ακριβής καθορισμός δεν είναι πάντοτε τόσο απλός όσο μπορεί να φαίνεται. Για παράδειγμα, οι δύο εναλλακτικές επιλογές που μόλις αναφέραμε για την *αντικατάσταση του μέγιστου* είναι αρκετά διαφορετικές: η πρώτη οδηγεί πάντοτε στην προσωρινή διόγκωση της ουράς κατά ένα στοιχείο, ενώ η δεύτερη τοποθετεί πάντοτε το νέο στοιχείο στην ουρά. Παρόμοια, η λειτουργία *αλλαγής προτεραιότητας* θα μπορούσε να υλοποιηθεί ως *διαγραφή* ακολουθούμενη από *εισαγωγή*, και η *κατασκευή* θα μπορούσε να υλοποιηθεί με επαναλαμβανόμενη χρήση της *εισαγωγής*.

Για μερικές εφαρμογές, μπορεί να είναι κάπως βολικότερο να χειριστούμε το *ελάχιστο* αντί για το μέγιστο κλειδί. Πάντως, κυρίως ασχολούμαστε με ουρές προτεραιότητας προσανατολισμένες στην προσπέλαση του μέγιστου κλειδιού. Όταν χρειαζόμαστε τον άλλο τύπο ουράς (μια ουρά προτεραιότητας που να μας επιτρέπει να *διαγράφουμε το ελάχιστο* στοιχείο), την αποκαλούμε *ελαχιστοστρεφή* (minimum-oriented) ουρά προτεραιότητας.

Η ουρά προτεραιότητας είναι ένας πρωτοτυπικός *αφηρημένος τύπος δεδομένων* (ΑΤΔ — δείτε το Κεφάλαιο 4): αντιπροσωπεύει ένα καλώς ορισμένο σύνολο λειτουργιών στα δεδομένα, και παρέχει μια κατάλληλη αφαίρεση που μας επιτρέπει να διακρίνουμε τα προγράμματα εφαρμογών (πελάτες) από τις διάφορες υλοποιήσεις που θα εξετάσουμε σε αυτό το κεφάλαιο. Στη διασύνδεση που παρουσιάζεται στο Πρόγραμμα 9.1 ορίζονται οι βασικότερες λειτουργίες της ουράς προτεραιότητας· θα δούμε μια πληρέστερη διασύνδεση στην Ενότητα 9.5. Αν θέλουμε να είμαστε απόλυτα ακριβείς, τα διαφορετικά υποσύνολα των διαφόρων λειτουργιών που είναι πιθανό να χρειαστεί να συμπεριλάβουμε οδηγούν σε διαφορετικές αφηρημένες δομές δεδομένων· ωστόσο, επειδή η ουρά προτεραιότητας χαρακτηρίζεται ουσιαστικά από τις λειτουργίες *διαγραφής του μέγιστου* και *εισαγωγής*, θα εστιάσουμε την ανάλυσή μας σε αυτές.

Οι διάφορες υλοποιήσεις των ουρών προτεραιότητας έχουν διαφορετικά χαρακτηριστικά επιδόσεων για τις διάφορες λειτουργίες που εκτελούν, και οι διάφορες εφαρμογές απαιτούν καλές επιδόσεις από τα διαφορετικά σύνολα λειτουργιών. Στην πραγματικότητα, οι διαφορές στις επιδόσεις είναι, κατά βάση, οι *μόνες* διαφορές που μπορούν να προκύψουν στην έννοια

Πρόγραμμα 9.1 Βασικός ΑΤΔ ουράς προτεραιότητας

Σε αυτή τη διασύνδεση ορίζονται λειτουργίες για τον απλούστερο τύπο ουράς προτεραιότητας: ανάθεση αρχικών τιμών, έλεγχος αν είναι κενή, προσθήκη νέου στοιχείου, διαγραφή του μεγαλύτερου στοιχείου. Οι στοιχειώδεις υλοποιήσεις αυτών των συναρτήσεων με τη χρήση πινάκων και συνδεδεμένων λιστών μπορεί να απαιτούν γραμμικό χρόνο στη χειρότερη περίπτωση, αλλά σε αυτό το κεφάλαιο θα συναντήσουμε υλοποιήσεις στις οποίες όλες οι λειτουργίες εκτελούνται εγγυημένα σε χρόνο το πολύ ανάλογο του λογαρίθμου του πλήθους των στοιχείων της ουράς. Το όρισμα της συνάρτησης `PQinit` καθορίζει το μέγιστο πλήθος των στοιχείων που αναμένονται στην ουρά.

```
void PQinit(int);
 int PQempty();
void PQinsert(Item);
Item PQdelmax();
```

του αφηρημένου τύπου δεδομένων. Αυτή η κατάσταση οδηγεί σε κάποιο αντίτιμο σε ό,τι αφορά το κόστος υλοποίησης. Σε αυτό το κεφάλαιο θα εξετάσουμε μια ποικιλία τρόπων προσέγγισης αυτού του αντίτιμου κόστους, μέχρι σχεδόν την ιδανική κατάσταση όπου μπορούμε να εκτελούμε τη λειτουργία *διαγραφής του μέγιστου* σε λογαριθμικό χρόνο και όλες τις άλλες λειτουργίες σε σταθερό χρόνο.

Καταρχάς, στην Ενότητα 9.1 θα παρουσιάσουμε αυτό το θέμα εξετάζοντας μερικές στοιχειώδεις δομές δεδομένων για την υλοποίηση των ουρών προτεραιότητας. Έπειτα, στις Ενότητες 9.2 έως 9.4 θα επικεντρώσουμε την προσοχή μας σε μια κλασική δομή δεδομένων που ονομάζεται *σωρός* (heap), η οποία επιτρέπει την αποδοτική υλοποίηση όλων των λειτουργιών εκτός από την *ένωση*. Στην Ενότητα 9.4 θα εξετάσουμε επίσης ένα σημαντικό αλγόριθμο ταξινόμησης που προκύπτει άμεσα από αυτές τις υλοποιήσεις. Στις Ενότητες 9.5 και 9.6 θα εξετάσουμε με περισσότερες λεπτομέρειες μερικά από τα προβλήματα που εμφανίζονται κατά την πλήρη ανάπτυξη ΑΤΔ ουρών προτεραιότητας. Τέλος, στην Ενότητα 9.7 θα μελετήσουμε μια πιο προηγμένη δομή δεδομένων, που ονομάζεται *διωνυμική ουρά* (binomial queue), την οποία χρησιμοποιούμε για να υλοποιήσουμε όλες τις λειτουργίες (μεταξύ των οποίων και την *ένωση*) σε λογαριθμικό χρόνο στη χειρότερη περίπτωση.

Κατά τη μελέτη όλων αυτών των διαφορετικών δομών δεδομένων, θα πρέπει να έχουμε κατά νου τόσο το βασικό αντίτιμο σε ό,τι αφορά το κόστος, το οποίο υπαγορεύεται από τη συνδεδεμένη ή την ακολουθιακή κατανομή μνήμης (όπως παρουσιάστηκαν στο Κεφάλαιο 3), όσο και τα προβλήματα που εμφανίζονται κατά τη δημιουργία πακέτων τα οποία χρησιμοποιούνται από τα προγράμματα εφαρμογών. Πιο συγκεκριμένα, μερικοί από τους προηγμένους αλγορίθμους που θα εξετάσουμε στη συνέχεια του βιβλίου είναι προγράμματα-πελάτες τα οποία χρησιμοποιούν ουρές προτεραιότητας.

Ασκήσεις

▷ **9.1** Στην παρακάτω ακολουθία, κάθε γράμμα σημαίνει *εισαγωγή* και κάθε αστερίσκος *διαγραφή του μέγιστου*:

P R I O * R * * I * T * Y * * * Q U E * * * U * E.

Παρουσιάστε τη σειρά των τιμών που επιστρέφονται από τις λειτουργίες *διαγραφής του μέγιστου*.

▷ **9.2** Προσθέστε στις συμβάσεις της Άσκησης 9.1 ένα σύμβολο πρόσθεσης που να σημαίνει *ένωση* και παρενθέσεις που να οριοθετούν την ουρά προτεραιότητας η οποία δημιουργείται από τις λειτουργίες που εκτελούνται μέσα στις παρενθέσεις. Δώστε τα περιεχόμενα της ουράς προτεραιότητας μετά την ακολουθία

(((P R I O *) + (R * I T * Y *)) * * *) + (Q U E * * * U * E).

○ **9.3** Εξηγήστε τον τρόπο χρήσης ενός ΑΤΔ ουράς προτεραιότητας όταν έχετε στόχο την υλοποίηση ενός ΑΤΔ στοίβας.

○ **9.4** Εξηγήστε τον τρόπο χρήσης ενός ΑΤΔ ουράς προτεραιότητας όταν έχετε στόχο την υλοποίηση ενός ΑΤΔ ουράς.

9.1 Στοιχειώδεις υλοποιήσεις

Οι βασικές δομές δεδομένων που γνωρίσαμε στο Κεφάλαιο 3 παρέχουν πολλές επιλογές για την υλοποίηση ουρών προτεραιότητας. Το Πρόγραμμα 9.2 είναι μια υλοποίηση στην οποία χρησιμοποιείται ένας αταξινόμητος πίνακας ως υποκείμενη δομή δεδομένων. Το πρόγραμμα υλοποιεί τη λειτουργία *εύρεσης του μέγιστου* σαρώνοντας τον πίνακα για την εύρεση του μέγιστου και αντιμεταθέτοντας, στη συνέχεια, το μέγιστο με το τελευταίο στοιχείο και μειώνοντας το μέγεθος της ουράς. Στην Εικόνα 9.1 παρουσιάζονται τα περιεχόμενα του πίνακα για ένα δείγμα ακολουθίας λειτουργιών. Η βασική υλοποίηση είναι αντίστοιχη με παρόμοιες υλοποιήσεις που είδαμε στο Κεφάλαιο 4 για στοίβες και ουρές (δείτε τα Προγράμματα 4.4 και 4.11) και είναι χρήσιμη για μικρές ουρές. Η σημαντική διαφορά βρίσκεται στην επίδοση. Για τις στοίβες και τις ουρές, είχαμε τη δυνατότητα να αναπτύσσουμε υλοποιήσεις όλων των λειτουργιών που απαιτούσαν σταθερό χρόνο· για τις ουρές προτεραιότητας, είναι εύκολο να βρούμε υλοποιήσεις όπου *είτε* ή συνάρτηση *εισαγωγής είτε* η συνάρτηση *διαγραφής του μέγιστου* απαιτεί σταθερό χρόνο, αλλά η εύρεση μιας υλοποίησης στην οποία να είναι γρήγορες *και οι δύο* λειτουργίες είναι δυσκολότερη και αποτελεί το θέμα αυτού του κεφαλαίου.

Πρόγραμμα 9.2 Υλοποίηση ουράς προτεραιότητας με πίνακα

Σε αυτή την υλοποίηση, η οποία είναι συγκρίσιμη με την υλοποίηση ουράς και στοίβας με τη χρήση πίνακα τις οποίες εξετάσαμε στο Κεφάλαιο 4 (δείτε το Πρόγραμμα 4.4), τα στοιχεία αποθηκεύονται σε ένα μη ταξινομημένο πίνακα. Τα στοιχεία προστίθενται και αφαιρούνται στο και από το τέλος του πίνακα με τον ίδιο τρόπο όπως και σε μια στοίβα.

```
#include <stdlib.h>
#include "Item.h"
static Item *pq;
static int N;
void PQinit(int maxN)
  { pq = malloc(maxN*sizeof(Item)); N = 0; }
 int PQempty()
  { return N == 0; }
void PQinsert(Item v)
  { pq[N++] = v; }
Item PQdelmax()
  { int j, max = 0;
    for (j = 1; j < N; j++)
      if (less(pq[max], pq[j])) max = j;
    exch(pq[max], pq[N-1]);
    return pq[--N];
  }
```

```
B         B
E         B E
*    E    B
S         B S
T         B S T
I         B S T I
*    T    B S I
N         L S I N
*    S    B N I
F         B N I F
I         B N I F I
R         B N I F I R
*    R    B N I F I
S         B N I F I S
T         B N I F I S T
*    T    B N I F I S
*    S    B N I F I
O         B N I F I O
U         B N I F I O U
*    U    B N I F I O
T         B N I F I O T
*    T    B N I F I O
*    O    B N I F I
*    N    B I I F
*    I    B F I
*    I    B F
*    F    B
*    B
```

Εικόνα 9.1
Παράδειγμα ουράς προτεραιότητας
(αναπαράσταση με μη ταξινομημένο πίνακα)

Σε αυτή την ακολουθία παρουσιάζεται το αποτέλεσμα της αλληλουχίας των λειτουργιών της αριστερής στήλης (από επάνω προς τα κάτω), όπου κάθε γράμμα σημαίνει εισαγωγή και κάθε αστερίσκος διαγραφή του μέγιστου. Σε κάθε γραμμή παρουσιάζεται η λειτουργία, το γράμμα που διαγράφεται από τις λειτουργίες διαγραφής του μέγιστου, και τα περιεχόμενα του πίνακα μετά τη λειτουργία.

Μπορούμε να χρησιμοποιήσουμε μη διατεταγμένες ή διατεταγμένες ακολουθίες υλοποιημένες ως συνδεδεμένες λίστες ή ως πίνακες. Το βασικό αντίτιμο ανάμεσα στο να αφήσουμε τα στοιχεία χωρίς ταξινόμηση και να τα έχουμε ταξινομημένα είναι ότι η διατήρηση μιας διατεταγμένης ακολουθίας επιτρέπει την εκτέλεση των λειτουργιών *διαγραφής του μέγιστου* και *εύρεσης του μέγιστου* σε σταθερό χρόνο, αλλά μπορεί να σημαίνει επίσης ότι για μια *εισαγωγή* θα πρέπει να διατρέξουμε ολόκληρη τη λίστα· αντίθετα, στην περίπτωση της μη διατεταγμένης ακολουθίας, η *εισαγωγή* γίνεται σε σταθερό χρόνο αλλά μπορεί να σημαίνει ότι για μια λειτουργία *διαγραφής του μέγιστου* και *εύρεσης του μέγιστου* θα χρειαστεί να διατρέξουμε ολόκληρη τη λίστα. Η μη διατεταγμένη ακολουθία αποτελεί την πρωτοτυπική *ράθυμη* (lazy) προσέγγιση σε αυτό το πρόβλημα, σύμφωνα με την οποία αναβάλλουμε κάποια εργασία μέχρι να είναι απαραίτητο να γίνει (στην περίπτωσή μας, την εύρεση του μέγιστου)· η διατεταγμένη ακολουθία αποτελεί την πρωτοτυπική *πρόθυμη* (eager) προσέγγιση στο πρόβλημα, σύμφωνα με την οποία κάνουμε όσο περισσότερη δουλειά μπορούμε από πριν (διατηρούμε τη λίστα ταξινομημένη σε κάθε εισαγωγή) ώστε να μπορούμε να εκτελέσουμε τις επόμενες λειτουργίες αποδοτικότερα. Μπορούμε να χρησιμοποιήσουμε αναπαράσταση πίνακα ή συνδεδεμένης λίστας και στις δύο περιπτώσεις, αλλά με δεδομένο ότι η (διπλά) συνδεδεμένη λίστα επιτρέπει *διαγραφή* σε σταθερό χρόνο (και, στην περίπτωση της μη ταξινομημένης περίπτωσης, *ένωση*), αλλά απαιτεί περισσότερο χώρο για τους συνδέσμους.

Στον Πίνακα 9.1 παρουσιάζεται το κόστος χειρότερης περίπτωσης των διαφόρων λειτουργιών (με προσέγγιση ενός σταθερού παράγοντα) σε μια ουρά προτεραιότητας μεγέθους N για διάφορες υλοποιήσεις.

Πίνακας 9.1 Κόστος χειρότερης περίπτωσης των λειτουργιών ουράς προτεραιότητας

Οι διάφορες υλοποιήσεις του αφηρημένου τύπου δεδομένων (ΑΤΔ) ουράς προτεραιότητας παρουσιάζουν μεγάλες διαφορές σε ό,τι αφορά τα χαρακτηριστικά επιδόσεών τους, κάτι που φαίνεται στον παρακάτω πίνακα όπου παρατίθεται ο χρόνος χειρότερης περίπτωσης των διαφόρων μεθόδων (με προσέγγιση ενός σταθερού συντελεστή για μεγάλο N). Οι στοιχειώδεις μέθοδοι (στις τέσσερις πρώτες γραμμές) απαιτούν σταθερό χρόνο για μερικές λειτουργίες και γραμμικό χρόνο για κάποιες άλλες, ενώ οι πιο προηγμένες μέθοδοι εγγυώνται επιδόσεις λογαριθμικού ή σταθερού χρόνου για τις περισσότερες από τις λειτουργίες.

	εισαγωγή	*διαγραφή μέγιστου*	*διαγραφή*	*εύρεση μέγιστου*	*αλλαγή προτεραιότητας*	*ένωση*
διατεταγμένος πίνακας	N	1	N	1	N	N
διατεταγμένη λίστα	N	1	1	1	N	N
μη διατεταγμένος πίνακας	1	N	1	N	1	N
μη διατεταγμένη λίστα	1	N	1	N	1	1
σωρός	$\lg N$	$\lg N$	$\lg N$	1	$\lg N$	N
διωνυμική ουρά	$\lg N$	$\lg N$	$\lg N$	$\lg N$	$\lg N$	$\lg N$
καλύτερος θεωρητικά	1	$\lg N$	$\lg N$	1	1	1

Η ανάπτυξη μιας πλήρους υλοποίησης απαιτεί την προσεκτική εξέταση της διασύνδεσης — ιδιαίτερα του τρόπου με τον οποίο τα προγράμματα-πελάτες προσπελάζουν τους κόμβους για τις λειτουργίες *διαγραφής* και *αλλαγής προτεραιότητας* και του τρόπου με τον οποίο προσεγγίζουν τις ίδιες τις ουρές προτεραιότητας ως τύπους δεδομένων για τη λειτουργία της *ένωσης*. Αυτά τα θέματα καλύπτονται στις Ενότητες 9.4 και 9.7, όπου δίνονται και δύο πλήρεις υλοποιήσεις — μία με διπλά συνδεδεμένες μη διατεταγμένες λίστες και μία με διωνυμικές ουρές.

Ο χρόνος εκτέλεσης ενός προγράμματος-πελάτη που χρησιμοποιεί ουρές προτεραιότητας εξαρτάται, όχι μόνο από τα κλειδιά, αλλά και από το μίγμα των διαφόρων λειτουργιών. Καλό είναι να μην ξεχνάμε τις απλές υλοποιήσεις επειδή, σε πολλές πρακτικές περιπτώσεις, συχνά μπορούν να ξεπεράσουν σε επιδόσεις τις πιο πολύπλοκες υλοποιήσεις. Για παράδειγμα, η υλοποίηση μη διατεταγμένης λίστας μπορεί να είναι κατάλληλη σε κάποια εφαρμογή όπου εκτελούνται μόνο λίγες λειτουργίες *διαγραφής του μέγιστου* αντί για πολλές εισαγωγές· αντίθετα, στην περίπτωση της εκτέλεσης πολλών λειτουργιών *εύρεσης του μέγιστου* ή στην περίπτωση που τα στοιχεία τα οποία εισάγονται είναι μεγαλύτερα από εκείνα που βρίσκονται ήδη στην ουρά προτεραιότητας, θα ήταν κατάλληλη μια διατεταγμένη λίστα.

Ασκήσεις

▷ **9.5** Κρίνετε την εξής ιδέα: για την υλοποίηση της *εύρεσης του μέγιστου* σε σταθερό χρόνο, γιατί να μην παρακολουθούμε τη μέγιστη τιμή που έχει εισαχθεί μέχρι εκείνη τη στιγμή και στη συνέχεια να επιστρέφουμε αυτή την τιμή για την *εύρεση του μέγιστου*;

▷ **9.6** Δώστε τα περιεχόμενα του πίνακα μετά την εκτέλεση της ακολουθίας λειτουργιών που φαίνονται στην Εικόνα 9.1.

9.7 Δώστε μια υλοποίηση για τη διασύνδεση της βασικής ουράς προτεραιότητας που να χρησιμοποιεί ως υποκείμενη δομή δεδομένων έναν ταξινομημένο πίνακα.

9.8 Δώστε μια υλοποίηση για τη διασύνδεση της βασικής ουράς προτεραιότητας που να χρησιμοποιεί ως υποκείμενη δομή δεδομένων μια μη διατεταγμένη συνδεδεμένη λίστα. *Υπόδειξη*: μελετήστε τα Προγράμματα 4.5 και 4.10.

9.9 Δώστε μια υλοποίηση για τη διασύνδεση της βασικής ουράς προτεραιότητας που να χρησιμοποιεί ως υποκείμενη δομή δεδομένων μια διατεταγμένη συνδεδεμένη λίστα. *Υπόδειξη*: μελετήστε το Πρόγραμμα 3.11.

○ **9.10** Θεωρήστε μια ράθυμη υλοποίηση στην οποία η λίστα ταξινομείται μόνο όταν εκτελείται μια λειτουργία *διαγραφής του μέγιστου* ή *εύρεσης του μέγιστου*. Οι εισαγωγές από την προηγούμενη ταξινόμηση αποθηκεύονται σε μια χωριστή λίστα και στη συνέχεια ταξινομούνται και συγχωνεύονται όποτε είναι απαραίτητο. Αναφέρετε τα πλεονεκτήματα μιας τέτοιας υλοποίησης σε σύγκριση με τις στοιχειώδεις υλοποιήσεις που βασίζονται σε μη διατεταγμένες και διατεταγμένες λίστες.

● **9.11** Γράψτε ένα πρόγραμμα-πελάτη που να χρησιμεύει ως πρόγραμμα οδήγησης επιδόσεων και να χρησιμοποιεί τη συνάρτηση `PQinsert` για να γεμίζει μια ουρά προτεραιότητας, έπειτα την `PQdelmax` για να αφαιρεί τα μισά κλειδιά, στη συνέχεια την `PQinsert` για να ξαναγεμίζει την ουρά, έπειτα την `PQdelmax` για να αφαιρεί όλα τα κλειδιά, και να εκτελεί αυτές τις λειτουργίες πολλές φορές σε τυχαίες ακολουθίες κλειδιών με διαφορετικά μήκη, τα οποία να κυμαίνονται από μικρά έως μεγάλα· να μετρά το χρόνο που χρειάζεται για κάθε εκτέλεση· και να εμφανίζει στην οθόνη ή να απεικονίζει σε διάγραμμα τους μέσους όρους των χρόνων εκτέλεσης.

● **9.12** Γράψτε ένα πρόγραμμα-πελάτη που να χρησιμεύει ως πρόγραμμα οδήγησης επιδόσεων και να χρησιμοποιεί τη συνάρτηση `PQinsert` για να γεμίζει μια ουρά προτεραιότητας, έπειτα να εκτελεί όσες περισσότερες φορές μπορεί τις `PQdelmax` και `PQinsert` μέσα σε 1 δευτερόλεπτο, και να εκτελεί αυτή τη σειρά λειτουργιών πολλές φορές σε τυχαίες ακολουθίες κλειδιών με διαφορετικά μήκη, τα οποία να κυμαίνονται από μικρά έως μεγάλα· και να εμφανίζει στην οθόνη ή να απεικονίζει σε διάγραμμα το μέσο πλήθος των λειτουργιών `PQdelmax` που κατάφερε να εκτελέσει.

9.13 Χρησιμοποιήστε το πρόγραμμα-πελάτη της Άσκησης 9.12 για να συγκρίνετε την υλοποίηση μη ταξινομημένου πίνακα του Προγράμματος 9.2 με την υλοποίηση μη διατεταγμένης λίστας της Άσκησης 9.8.

9.14 Χρησιμοποιήστε το πρόγραμμα-πελάτη της Άσκησης 9.12 για να συγκρίνετε την υλοποίηση διατεταγμένου πίνακα με την υλοποίηση διατεταγμένης λίστας των Ασκήσεων 9.7 και 9.9.

● **9.15** Γράψτε ένα πρόγραμμα-πελάτη που να χρησιμεύει ως πρόγραμμα οδήγησης άσκησης και να χρησιμοποιεί τις συναρτήσεις της διασύνδεσης ουράς προτεραιότητας του Προγράμματος 9.1 σε δύσκολες ή "παθολογικές" περιπτώσεις οι οποίες ενδέχεται να παρουσιαστούν σε πρακτικές εφαρμογές. Απλά παραδείγματα περιλαμβάνουν κλειδιά που είναι ήδη ταξινομημένα, κλειδιά με αντίστροφη ταξινόμηση, κλειδιά που είναι όλα ίδια μεταξύ τους, και ακολουθίες κλειδιών που έχουν μόνο δύο διακεκριμένες τιμές.

9.16 (Αυτή η άσκηση είναι στην ουσία 24 ασκήσεις.) Δικαιολογήστε τα φράγματα χειρότερης περίπτωσης για τις τέσσερις στοιχειώδεις υλοποιήσεις που δίνονται στον Πίνακα 9.1 σε σχέση με την υλοποίηση του Προγράμματος 9.2 και τις υλοποιήσεις που δημιουργήσατε στις Ασκήσεις 9.7 έως και 9.9 για τις λειτουργίες *εισαγωγής* και *διαγραφής του μέγιστου*, και περιγράφοντας ανεπίσημα τις μεθόδους για τις άλλες λειτουργίες. Για τη *διαγραφή*, την *αλλαγή προτεραιότητας*, και την *ένωση*, θεωρήστε ότι έχετε κάποιο χειριστήριο που σας παρέχει άμεση πρόσβαση στο αντικείμενο της αναφοράς.

9.2 Δομή δεδομένων σωρού

Το κύριο θέμα αυτού του κεφαλαίου είναι μια απλή δομή δεδομένων που ονομάζεται *σωρός* (heap) και μπορεί να υποστηρίζει αποδοτικά τις βασικές λειτουργίες μιας ουράς προτεραιότητας. Σε ένα σωρό, οι εγγραφές αποθηκεύονται σε έναν πίνακα έτσι ώστε κάθε κλειδί να είναι εγγυημένα μεγαλύτερο από τα κλειδιά που βρίσκονται σε δύο άλλες συγκεκριμένες θέσεις. Με τη σειρά τους, καθένα από αυτά τα κλειδιά πρέπει να είναι μεγαλύτερο από δύο ακόμη κλειδιά, και ούτω καθεξής. Αυτή η διάταξη γίνεται αρκετά ευδιάκριτη αν φανταστούμε ότι τα κλειδιά βρίσκονται σε κάποια δομή δυαδικού δένδρου με ακμές από κάθε κλειδί προς τα δύο κλειδιά τα οποία είναι γνωστό ότι είναι μικρότερα από αυτό.

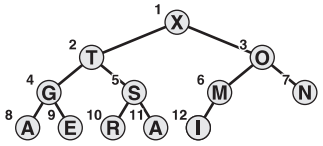
Ορισμός 9.2 *Ένα δένδρο είναι* **διατεταγμένο σε σωρό** *(heap-ordered) αν το κλειδί που περιέχεται σε κάθε κόμβο είναι μεγαλύτερο ή ίσο με τα κλειδιά όλων των παιδιών αυτού του κόμβου (αν βέβαια υπάρχουν). Ισοδύναμα, το κλειδί που περιέχεται σε κάθε κόμβο ενός δένδρου το οποίο είναι διατεταγμένο σε σωρό είναι μικρότερο ή ίσο από το κλειδί του γονέα αυτού του κόμβου (αν βέβαια υπάρχει).*

Ιδιότητα 9.1 *Κανένας κόμβος ενός δένδρου που είναι διατεταγμένο σε σωρό δεν έχει κλειδί μεγαλύτερο από το κλειδί του κόμβου της ρίζας.*

Θα μπορούσαμε να επιβάλουμε τον περιορισμό της διάταξης σε σωρό σε όλα τα δένδρα, αλλά είναι ιδιαίτερα βολικό να χρησιμοποιήσουμε ένα *πλήρες δυαδικό δένδρο* (complete binary tree). Θυμηθείτε από το Κεφάλαιο 3 ότι μπορούμε να σχεδιάσουμε μια τέτοια δομή τοποθετώντας τον κόμβο της ρίζας και προχωρώντας στη συνέχεια προς τα κάτω, και από τα αριστερά προς τα δεξιά, ενώνοντας δύο κόμβους κάτω από κάθε κόμβο του προηγούμενου επιπέδου μέχρι να τοποθετηθούν N κόμβοι. Μπορούμε να αναπαραστήσουμε ένα πλήρες δυαδικό δένδρο ακολουθιακά σε έναν πίνακα, τοποθετώντας απλώς τη ρίζα στη θέση 1, τα παιδιά της στις θέσεις 2 και 3, τους κόμβους του επόμενου επιπέδου στις θέσεις 4, 5, 6, και 7, και ούτω καθεξής, όπως φαίνεται στην Εικόνα 9.2. ■

Ορισμός 9.3 *Ο* **σωρός** *(heap) είναι ένα σύνολο κόμβων με κλειδιά τοποθετημένα σε ένα πλήρες δυαδικό δένδρο το οποίο είναι διατεταγμένο σε σωρό και αναπαρίσταται ως πίνακας.*

Θα μπορούσαμε να χρησιμοποιήσουμε μια συνδεδεμένη αναπαράσταση για δένδρα διατεταγμένα σε σωρό, αλλά τα πλήρη δένδρα μάς επιτρέπουν να χρησιμοποιήσουμε μια συμπαγή αναπαράσταση πίνακα όπου μπορούμε εύκολα να μεταφερθούμε από έναν κόμβο στο γονέα του και στα παιδιά του χωρίς να χρειάζεται να διατηρούμε άμεσους συνδέσμους. Ο γονέας του κόμβου στη θέση i βρίσκεται στη θέση $\lfloor i/2 \rfloor$ και, αντίστροφα, τα δύο παιδιά του κόμβου στη θέση i βρίσκονται στις θέσεις $2i$ και $2i+1$. Αυτή η διάταξη κάνει τη διάσχιση μέσα από



Εικόνα 9.2
Αναπαράσταση πίνακα ενός πλήρους δυαδικού δένδρου διατεταγμένου σε σωρό

Η θεώρηση ότι το στοιχείο που βρίσκεται στη θέση $\lfloor i/2 \rfloor$ ενός πίνακα είναι ο γονέας του στοιχείου που βρίσκεται στη θέση i, για $2 \le i \le N$ (ή ισοδύναμα, ότι το i-οστό στοιχείο είναι γονέας του $2i$-οστού στοιχείου και του $(2i+1)$-οστού στοιχείου), αντιστοιχεί σε μια κατάλληλη αναπαράσταση των στοιχείων ενός δένδρου. Αυτή η αντιστοιχία είναι ισοδύναμη με την αρίθμηση των κόμβων ενός πλήρους δυαδικού δένδρου (με κόμβους στο κατώτατο επίπεδο και όσο το δυνατόν αριστερότερα) κατά επίπεδα. Ένα δένδρο είναι διατεταγμένο σε σωρό αν το κλειδί σε οποιονδήποτε δεδομένο κόμβο είναι μεγαλύτερο ή ίσο από τα κλειδιά των παιδιών αυτού του κόμβου. Ο σωρός είναι μια αναπαράσταση πίνακα ενός πλήρους δυαδικού δένδρου διατεταγμένου σε σωρό. Το i-οστό στοιχείο ενός σωρού είναι μεγαλύτερο ή ίσο τόσο από το $2i$-οστό στοιχείο όσο και από το $(2i+1)$-οστό στοιχείο.

ένα τέτοιο δένδρο ακόμη πιο εύκολη σε σύγκριση με την περίπτωση που το δένδρο έχει υλοποιηθεί με συνδεδεμένη αναπαράσταση, επειδή σε μια συνδεδεμένη αναπαράσταση θα έπρεπε να είχαμε τρεις συνδέσμους συσχετισμένους με κάθε κλειδί, έτσι ώστε να μπορούμε να μετακινούμαστε προς τα επάνω και προς τα κάτω μέσα στο δένδρο (κάθε στοιχείο θα είχε ένα δείκτη προς το γονέα του και ένα δείκτη προς το κάθε παιδί του). Τα πλήρη δυαδικά δένδρα που αναπαρίστανται ως πίνακες είναι ανθεκτικές δομές, αλλά διαθέτουν και την αναγκαία ευελιξία ώστε να μας επιτρέπουν να υλοποιούμε αποδοτικούς αλγορίθμους ουρών προτεραιότητας.

Στην Ενότητα 9.3 θα δούμε ότι μπορούμε να χρησιμοποιούμε σωρούς για την υλοποίηση όλων των λειτουργιών ουράς προτεραιότητας (εκτός από την *ένωση*), με τέτοιον τρόπο ώστε να απαιτούν λογαριθμικούς χρόνους στη χειρότερη περίπτωση. Όλες οι υλοποιήσεις λειτουργούν κατά μήκος μιας διαδρομής μέσα στο σωρό (η διέλευση γίνεται από το γονέα προς το παιδί και προς τα κάτω, ή από το παιδί προς το γονέα και προς τα επάνω, αλλά χωρίς αλλαγή κατεύθυνσης). Όπως αναφέρθηκε στο Κεφάλαιο 3, όλες οι διαδρομές σε ένα πλήρες δένδρο N κόμβων έχουν περίπου $\lg N$ κόμβους — υπάρχουν περίπου $N/2$ κόμβοι στο κάτω μέρος, $N/4$ κόμβοι με παιδιά στο κάτω μέρος, $N/8$ κόμβοι με εγγόνια στο κάτω μέρος, και ούτω καθεξής. Κάθε γενιά έχει περίπου τους μισούς κόμβους από την επόμενη, και υπάρχουν το πολύ $\lg N$ γενιές.

Μπορούμε επίσης να χρησιμοποιήσουμε ρητές συνδεδεμένες αναπαραστάσεις δομών δένδρου για να αναπτύξουμε αποδοτικές υλοποιήσεις των λειτουργιών ουράς προτεραιότητας. Παραδείγματα αποτελούν τα τριπλά συνδεδεμένα πλήρη δένδρα που είναι διατεταγμένα σε σωρό (δείτε την Ενότητα 9.5), τα τουρνουά (δείτε το Πρόγραμμα 5.19), και οι διωνυμικές ουρές (δείτε την Ενότητα 9.7). Όπως και στις απλές στοίβες και ουρές, ένας βασικός λόγος για να εξετάσουμε συνδεδεμένες αναπαραστάσεις είναι ότι μας αποδεσμεύουν από την υποχρέωση να γνωρίζουμε από πριν το μέγιστο μέγεθος της ουράς, όπως απαιτείται στην αναπαράσταση πίνακα. Σε συγκεκριμένες καταστάσεις, μπορούμε επίσης να αξιοποιήσουμε την ευελιξία που παρέχουν οι συνδεδεμένες δομές ώστε να αναπτύξουμε αποδοτικούς αλγορίθμους. Οι αναγνώστες που δεν έχουν πείρα στη χρήση ρητών δομών δένδρου μπορούν να μελετήσουν το Κεφάλαιο 12 προκειμένου να μάθουν τις βασικές μεθόδους για την ακόμη σημαντικότερη υλοποίηση του ΑΤΔ πίνακα συμβόλων, πριν ασχοληθούν με τις συνδεδεμένες

αναπαραστάσεις δένδρου στις ασκήσεις αυτού του κεφαλαίου και στην Ενότητα 9.7. Παρόλα αυτά, μπορείτε να αναβάλετε την προσεκτικότερη μελέτη των συνδεδεμένων δομών και να την κάνετε σε μια δεύτερη ανάγνωση, επειδή το κύριο θέμα αυτού του κεφαλαίου είναι ο σωρός (αναπαράσταση πίνακα χωρίς συνδέσμους για πλήρες δένδρο διατεταγμένο σε σωρό).

Ασκήσεις

▷ **9.17** Ένας πίνακας ταξινομημένος σε φθίνουσα σειρά είναι δομή σωρού;

○ **9.18** Το μεγαλύτερο στοιχείο ενός σωρού πρέπει να βρίσκεται στη θέση 1 και το επόμενο μεγαλύτερο στοιχείο πρέπει να βρίσκεται στη θέση 2 ή στη θέση 3. Για ένα σωρό μεγέθους 15, δώστε τη λίστα των θέσεων όπου το k-οστό μεγαλύτερο στοιχείο (α) μπορεί να υπάρχει και (β) δεν μπορεί να υπάρχει, για $k = 2, 3, 4$ (θεωρώντας τις τιμές διακεκριμένες).

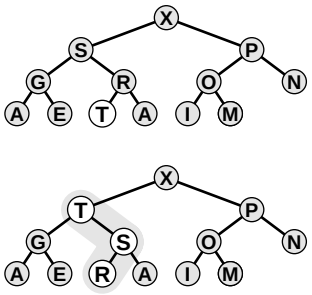
● **9.19** Λύστε την Άσκηση 9.18 γενικά για k ως συνάρτηση του N, όπου N το μέγεθος του σωρού.

● **9.20** Λύστε τις Ασκήσεις 9.18 και 9.19 για το k-οστό *μικρότερο* στοιχείο.

9.3 Αλγόριθμοι σε σωρούς

Όλοι οι αλγόριθμοι ουρών προτεραιότητας σε σωρούς λειτουργούν κάνοντας αρχικά μια απλή τροποποίηση που θα μπορούσε να παραβιάσει τη συνθήκη σωρού, και στη συνέχεια διατρέχουν το σωρό τροποποιώντας τον όπως απαιτείται προκειμένου να διασφαλιστεί ότι η συνθήκη σωρού ικανοποιείται παντού. Αυτή η διαδικασία ονομάζεται μερικές φορές *τακτοποίηση σωρού* (heapifying). Υπάρχουν δύο περιπτώσεις. Όταν η προτεραιότητα ενός κόμβου είναι αυξημένη (ή προστίθεται ένας νέος κόμβος στο κάτω μέρος του σωρού), για να αποκαταστήσουμε τη συνθήκη του σωρού πρέπει να μετακινηθούμε *προς τα επάνω* μέσα στο σωρό. Όταν η προτεραιότητα ενός κόμβου είναι μειωμένη (για παράδειγμα, αν αντικαταστήσουμε τον κόμβο της ρίζας με ένα νέο κόμβο), για να αποκαταστήσουμε τη συνθήκη του σωρού πρέπει να μετακινηθούμε *προς τα κάτω* μέσα στο σωρό. Θα εξετάσουμε πρώτα τον τρόπο με τον οποίο πρέπει να υλοποιήσουμε αυτές τις δύο βασικές συναρτήσεις, και στη συνέχεια θα δούμε πώς πρέπει να τις χρησιμοποιήσουμε για να υλοποιήσουμε τις διάφορες λειτουργίες ουράς προτεραιότητας.

Αν παραβιαστεί η ιδιότητα του σωρού επειδή το κλειδί κάποιου κόμβου έγινε μεγαλύτερο από το κλειδί του γονικού του κόμβου, μπορούμε να αποκαταστήσουμε την ιδιότητα αντιμεταθέτοντας τον κόμβο με το γονέα του. Μετά την αντιμετάθεση, ο κόμβος θα είναι μεγαλύτερος και από τα δύο παιδιά του (το ένα παιδί θα είναι ο παλιός γονέας και το άλλο θα είναι μικρότερο από τον παλιό γονέα επειδή ήταν παιδί αυτού του κόμβου) αλλά μπορεί να εξακολουθεί να είναι μεγαλύτερος από το γονέα του. Μπορούμε να ρυθμίσουμε αυτή την ασυμφωνία με τον ίδιο τρόπο, και ούτω καθεξής, ανεβαίνοντας μέσα στο σωρό μέχρι να φτάσουμε είτε σε κάποιον κόμβο με μεγαλύτερο κλειδί είτε στη ρίζα. Ένα παράδειγμα αυτής της διαδικασίας φαίνεται στην Εικόνα 9.3. Ο κώδικας είναι απλός και βασίζεται στην έννοια ότι ο γονέας του κόμβου, που βρίσκεται στη θέση `k` ενός σωρού, είναι στη θέση `k/2`. Το Πρόγραμμα 9.3 αποτελεί υλοποίηση μιας συνάρτησης που διορθώνει μια πιθανή παραβίαση αύξησης της προτεραιότητας ενός δοθέντος κόμβου κάποιου σωρού, ανεβαίνοντας μέσα στο σωρό.



Εικόνα 9.3
Συνθετική τακτοποίηση σωρού

Το επάνω δένδρο είναι διατεταγμένο σε σωρό εκτός από τον κόμβο T *στο κατώτατο επίπεδο. Αν αντιμεταθέσουμε τον κόμβο* T *με το γονέα του, το δένδρο θα είναι διατεταγμένο σε σωρό, εκτός ίσως από τον κόμβο* T *ο οποίος μπορεί να εξακολουθεί να είναι μεγαλύτερος από το γονέα του. Εξακολουθούμε να αντιμεταθέτουμε τον κόμβο* T *με το γονέα του μέχρι να φτάσουμε στη ρίζα ή σε κάποιον κόμβο στη διαδρομή από τον* T *προς τη ρίζα που να είναι μεγαλύτερος από τον* T*, έτσι ώστε να αποκαταστήσουμε τη συνθήκη του σωρού σε όλο το δένδρο. Μπορούμε να χρησιμοποιήσουμε αυτή τη διαδικασία ως βάση για τη λειτουργία εισαγωγής ενός στοιχείου σε σωρό, έτσι ώστε να αποκαθιστούμε τη συνθήκη του σωρού αφού προσθέσουμε ένα νέο στοιχείο σε αυτόν (στη δεξιότερη θέση στο κατώτατο επίπεδο, δημιουργώντας και νέο επίπεδο αν είναι απαραίτητο).*

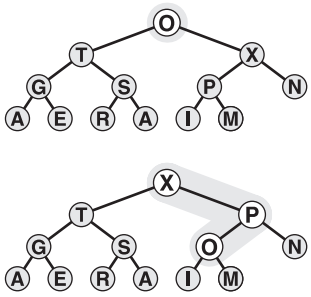
Πρόγραμμα 9.3 Συνθετική (bottom-up) τακτοποίηση σωρού

Για την αποκατάσταση της συνθήκης του σωρού, όταν αυξηθεί η προτεραιότητα ενός κόμβου, ανεβαίνουμε μέσα στο σωρό αντιμεταθέτοντας, αν είναι απαραίτητο, τον κόμβο στη θέση `k` με το γονέα του (στη θέση `k/2`) και συνεχίζοντας με αυτόν τον τρόπο όσο `a[k/2]<a[k]` ή μέχρι να καταλήξουμε στην κορυφή του σωρού.

```
fixUp(Item a[], int k)
  {
    while (k > 1 && less(a[k/2], a[k]))
      { exch(a[k], a[k/2]); k = k/2; }
  }
```

Αν παραβιαστεί η ιδιότητα του σωρού επειδή το κλειδί ενός κόμβου γίνει μικρότερο από το κλειδί του κόμβου τού ενός ή και των δύο παιδιών του, τότε μπορούμε να εξαλείψουμε την παραβίαση αντιμεταθέτοντας τον κόμβο με το μεγαλύτερο από τα δύο παιδιά του. Αυτή η αντιμετάθεση μπορεί να προκαλέσει παραβίαση στη θέση που βρίσκεται το παιδί· διορθώνουμε και αυτή την παραβίαση με τον ίδιο τρόπο, και ούτω καθεξής, κατεβαίνοντας μέσα στο σωρό μέχρι να καταλήξουμε είτε σε κάποιον κόμβο του οποίου και τα δύο παιδιά είναι μικρότερα είτε στη βάση του σωρού. Ένα παράδειγμα αυτής της διαδικασίας φαίνεται στην Εικόνα 9.4. Ο κώδικας προκύπτει άμεσα από το γεγονός ότι τα παιδιά ενός κόμβου, ο οποίος βρίσκεται στη θέση `k` ενός σωρού, είναι στις θέσεις `2k` και `2k+1`. Το Πρόγραμμα 9.4 αποτελεί υλοποίηση μιας συνάρτησης που διορθώνει μια πιθανή παραβίαση εξαιτίας αύξησης της προτεραιότητας ενός δοθέντος κόμβου ενός σωρού, κατεβαίνοντας μέσα στο σωρό. Αυτή η συνάρτηση χρειάζεται να γνωρίζει το μέγεθος του σωρού (`N`), έτσι ώστε να μπορεί να ελέγχει μήπως έχει φτάσει στη βάση του σωρού.

Αυτές οι δύο λειτουργίες είναι ανεξάρτητες από τον τρόπο αναπαράστασης της δομής δένδρου, με την προϋπόθεση ότι μπορούμε να προσπελάζουμε το γονέα (για τη συνθετική μέθοδο) και τα παιδιά (για την αναλυτική μέθοδο) κάθε κόμβου. Στη συνθετική μέθοδο, ανεβαίνουμε μέσα στο δένδρο αντιμεταθέτοντας το κλειδί του δεδομένου κόμβου με το κλειδί του γονέα του, μέχρι να φτάσουμε στη ρίζα ή σε κάποιο γονέα με κλειδί μεγαλύτερο (ή ίσο).



Εικόνα 9.4
Αναλυτική τακτοποίηση σωρού

Το επάνω δένδρο είναι διατεταγμένο σε σωρό εκτός από τον κόμβο που βρίσκεται στη ρίζα. Αν αντιμεταθέσουμε το O *με το μεγαλύτερο από τα δύο παιδιά του (*X*), το δένδρο θα είναι διατεταγμένο σε σωρό εκτός ίσως από το υποδένδρο που έχει ρίζα τον κόμβο* O*. Συνεχίζουμε να αντιμεταθέτουμε τον κόμβο* O *με το μεγαλύτερο από τα δύο παιδιά του μέχρι το κατώτατο επίπεδο του σωρού, ή κάποιο σημείο όπου το* O *είναι μεγαλύτερο και από τα δύο παιδιά του, έτσι ώστε να αποκαταστήσουμε τη συνθήκη του σωρού σε όλο το δένδρο. Μπορούμε να χρησιμοποιήσουμε αυτή τη διαδικασία ως βάση για τη λειτουργία διαγραφής του μεγαλύτερου στοιχείου από ένα σωρό, έτσι ώστε να αποκαθιστούμε τη συνθήκη του σωρού αφού αντικαταστήσουμε το κλειδί στη ρίζα με το κλειδί που βρίσκεται στη δεξιότερη θέση του κατώτατου επιπέδου.*

Πρόγραμμα 9.4 Αναλυτική (top-down) τακτοποίηση σωρού

Για την αποκατάσταση της συνθήκης σωρού όταν μειωθεί η προτεραιότητα κάποιου κόμβου κατεβαίνουμε μέσα στο σωρό αντιμεταθέτοντας, αν είναι απαραίτητο, τον κόμβο που βρίσκεται στη θέση `k` με το μεγαλύτερο από τα παιδιά του και σταματώντας όταν ο κόμβος στη θέση `k` δεν είναι μικρότερος και από τα δύο παιδιά του ή όταν καταλήξουμε στη βάση του σωρού. Παρατηρήστε ότι, αν το `N` είναι άρτιος και το `k` είναι `N/2`, ο κόμβος στο `k` θα έχει μόνο ένα παιδί — αυτή την περίπτωση πρέπει να τη χειριστούμε προσεκτικά!

Ο εσωτερικός βρόχος αυτού του προγράμματος έχει δύο διακεκριμένες εξόδους: μία για την περίπτωση που θα φτάσουμε στη βάση του σωρού και μία για την περίπτωση που η συνθήκη του σωρού θα ικανοποιηθεί στο εσωτερικό του. Πρόκειται για ένα πρωτοτυπικό παράδειγμα όπου είναι απαραίτητη η χρήση της εντολής `break`.

```
fixDown(Item a[], int k, int N)
  { int j;
    while (2*k <= N)
      { j = 2*k;
        if (j < N && less(a[j], a[j+1])) j++;
        if (!less(a[k], a[j])) break;
        exch(a[k], a[j]); k = j;
      }
  }
```

Στην αναλυτική μέθοδο, κατεβαίνουμε μέσα στο δένδρο αντιμεταθέτοντας το κλειδί του δεδομένου κόμβου με το μεγαλύτερο κλειδί των παιδιών αυτού του κόμβου, κατεβαίνοντας σε εκείνο το παιδί, και συνεχίζοντας μέχρι να φτάσουμε στη βάση του δένδρου ή σε ένα σημείο όπου κανένα από τα παιδιά δεν έχει μεγαλύτερο κλειδί. Γενικευμένες με αυτόν τον τρόπο, οι συγκεκριμένες λειτουργίες εφαρμόζονται, όχι μόνο σε πλήρη δυαδικά δένδρα αλλά και σε οποιαδήποτε άλλη δομή δένδρου. Οι προηγμένοι αλγόριθμοι ουρών προτεραιότητας χρησιμοποιούν συνήθως γενικότερες δομές δένδρου, αλλά βασίζονται στις ίδιες αυτές βασικές λειτουργίες προκειμένου να διατηρούν την πρόσβαση στο μεγαλύτερο κλειδί της δομής, το οποίο βρίσκεται στην κορυφή.

Αν φανταστούμε ότι ο σωρός αναπαριστά την ιεραρχία ενός οργανισμού, όπου κάθε παιδί ενός κόμβου αναπαριστά υφισταμένους (και ο γονέας αναπαριστά τον άμεσο προϊστάμενό τους), αυτές οι λειτουργίες έχουν ενδιαφέρουσες ερμηνείες. Η συνθετική μέθοδος αντιστοιχεί σε έναν πολλά υποσχόμενο νέο μάνατζερ που εμφανίζεται στη σκηνή, ο οποίος προβιβάζεται στην αλυσίδα της ιεραρχίας (ανταλλάσσοντας θέσεις εργασίας με προϊσταμένους που διαθέτουν λιγότερες ικανότητες) μέχρι να συναντήσει έναν προϊστάμενο με περισσότερες ικανότητες. Η αναλυτική μέθοδος είναι ανάλογη με την περίπτωση που ο πρόεδρος της εταιρείας αντικαθίσταται από κάποιον με λιγότερα προσόντα. Αν ο ισχυρότερος υφιστάμενος του προέδρου είναι ισχυρότερος από αυτό το νέο πρόσωπο, ανταλλάσσουν θέσεις εργασίας και κατεβαίνουμε την αλυσίδα της ιεραρχίας, υποβιβάζοντας το νέο πρόσωπο και προβιβάζοντας άλλους μέχρι να φτάσουμε στο επίπεδο των προσόντων αυτού του προσώπου, όπου δεν υπάρχει υφιστάμενος με περισσότερα προσόντα (σπάνια, βέβαια, βλέπουμε αυτό το ιδανικό σενάριο στον πραγματικό κόσμο). Βασιζόμενοι σε αυτή την αναλογία, συχνά αναφερόμαστε στην προς τα επάνω μετακίνηση μέσα σε ένα σωρό με τον όρο *προαγωγή* (promotion).

Αυτές οι δύο βασικές λειτουργίες επιτρέπουν την αποδοτική υλοποίηση του βασικού ΑΤΔ ουράς προτεραιότητας, όπως φαίνεται στο Πρόγραμμα 9.5. Με την ουρά προτεραιότητας ως πίνακα διατεταγμένο σε σωρό, η χρήση της λειτουργίας *εισαγωγής* (insert) αντιστοιχεί στην προσθήκη του νέου στοιχείου στο τέλος και τη μετακίνηση αυτού του στοιχείου προς τα

Πρόγραμμα 9.5 Ουρά προτεραιότητας βασισμένη σε σωρό

Για την υλοποίηση της συνάρτησης `PQinsert`, αυξάνουμε το `N` κατά 1, προσθέτουμε το νέο στοιχείο στο τέλος του σωρού, και χρησιμοποιούμε τη συνάρτηση `fixUp` για να αποκαταστήσουμε τη συνθήκη του σωρού. Στην περίπτωση της συνάρτησης `PQdelmax`, το μέγεθος του σωρού πρέπει να μειωθεί κατά 1, οπότε παίρνουμε την τιμή που πρέπει να επιστραφεί από το `pq[1]` και μειώνουμε το μέγεθος του σωρού μετακινώντας το στοιχείο `pq[N]` στο `pq[1]` και χρησιμοποιώντας τη συνάρτηση `fixDown` προκειμένου να αποκαταστήσουμε τη συνθήκη του σωρού. Η υλοποίηση των συναρτήσεων `PQinit` και `PQempty` είναι πολύ εύκολη. Η πρώτη θέση του πίνακα, `pq[0]`, δεν χρησιμοποιείται, αλλά ενδέχεται να είναι διαθέσιμη ως φρουρός σε κάποιες υλοποιήσεις.

```
#include <stdlib.h>
#include "Item.h"
static Item *pq;
static int N;
void PQinit(int maxN)
  { pq = malloc((maxN+1)*sizeof(Item)); N = 0; }
 int PQempty()
  { return N == 0; }
void PQinsert(Item v)
  { pq[++N] = v; fixUp(pq, N); }
Item PQdelmax()
  {
    exch(pq[1], pq[N]);
    fixDown(pq, 1, N-1);
    return pq[N--];
  }
```

επάνω μέσα στο σωρό, προκειμένου να αποκατασταθεί η συνθήκη του σωρού· η λειτουργία *διαγραφής του μέγιστου* αντιστοιχεί στην αφαίρεση της μεγαλύτερης τιμής από την κορυφή, και στη συνέχεια την τοποθέτηση στην κορυφή του στοιχείου που βρίσκεται στο τέλος του σωρού και τη μετακίνησή του προς τα κάτω μέσα στον πίνακα, προκειμένου να αποκατασταθεί η συνθήκη του σωρού.

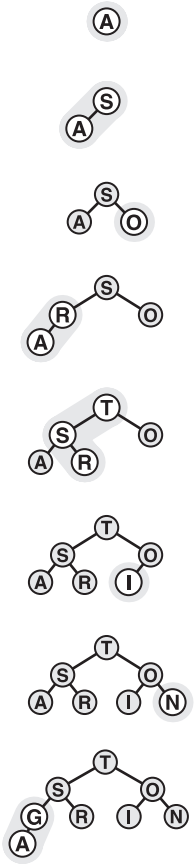
Ιδιότητα 9.2 *Οι λειτουργίες* **εισαγωγής** *και* **διαγραφής του μέγιστου** *για τον αφηρημένο τύπο δεδομένων ουράς προτεραιότητας μπορούν να υλοποιηθούν με δένδρα διατεταγμένα σε σωρό, έτσι ώστε, όταν εφαρμόζονται σε μια ουρά N στοιχείων, η* **εισαγωγή** *να απαιτεί το πολύ* $\lg N$ *συγκρίσεις και η* **διαγραφή του μέγιστου** *να απαιτεί το πολύ* $2 \lg N$ *συγκρίσεις.*

Και οι δύο λειτουργίες περιλαμβάνουν μετακίνηση κατά μήκος μιας διαδρομής ανάμεσα στη ρίζα και τη βάση του σωρού, και καμία διαδρομή ενός σωρού μεγέθους N δεν περιλαμβάνει περισσότερα από $\lg N$ στοιχεία (δείτε, για παράδειγμα, την Ιδιότητα 5.8 και την Άσκηση 5.77). Η λειτουργία *διαγραφής του μέγιστου* απαιτεί δύο συγκρίσεις για κάθε κόμβο — μία για να βρεθεί το παιδί με το μεγαλύτερο κλειδί και μια άλλη για να αποφασιστεί αν αυτό το παιδί πρέπει να προαχθεί. ■

Στις Εικόνες 9.5 και 9.6 παρουσιάζεται ένα παράδειγμα στο οποίο κατασκευάζουμε ένα σωρό εισάγοντας στοιχεία το ένα μετά το άλλο σε ένα σωρό που ήταν αρχικά άδειος. Στην αναπαράσταση πίνακα που χρησιμοποιήσαμε για το σωρό, διατάξαμε τον πίνακα σε σωρό μετακινούμενοι ακολουθιακά μέσα στον πίνακα, θεωρώντας ότι το μέγεθος του σωρού αυξάνεται κατά 1 κάθε φορά που μετακινούμαστε σε ένα νέο στοιχείο, και χρησιμοποιώντας τη συνάρτηση `fixUp` για να αποκαταστήσουμε τη διάταξη του σωρού. Η διαδικασία απαιτεί στη χειρότερη περίπτωση χρόνο ανάλογο του $N \log N$ (αν κάθε νέο στοιχείο είναι το μεγαλύτερο που έχει εμφανιστεί μέχρι εκείνη τη στιγμή, η διαδικασία διατρέχει όλο το σωρό μέχρι την κορυφή του), αλλά τελικά προκύπτει ότι απαιτείται μόνο γραμμικός χρόνος κατά μέσο όρο (κάθε τυχαίο νέο στοιχείο συνήθως μετακινείται προς τα επάνω μόνο κατά λίγα επίπεδα). Στην Ενότητα 9.4 θα δούμε έναν τρόπο κατασκευής σωρού (προκειμένου να διατάσσουμε έναν πίνακα σε σωρό) ο οποίος απαιτεί γραμμικό χρόνο στη χειρότερη περίπτωση.

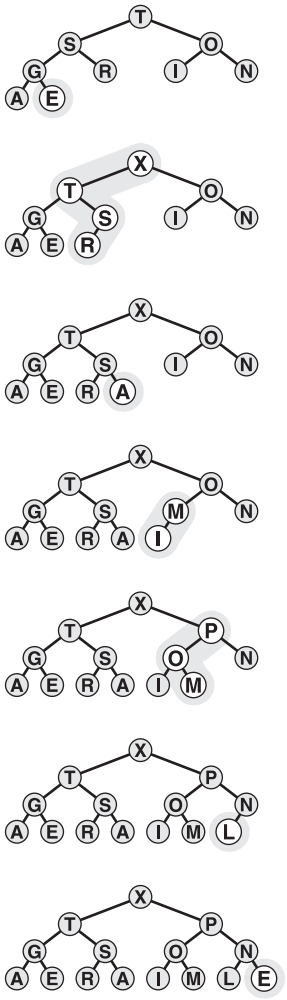
Οι βασικές λειτουργίες `fixUp` και `fixDown` των Προγραμμάτων 9.3 και 9.4 επιτρέπουν επίσης την άμεση υλοποίηση των λειτουργιών *αλλαγής προτεραιότητας* και *διαγραφής*. Για να αλλάξουμε την προτεραιότητα ενός στοιχείου που βρίσκεται κάπου στο μέσο του σωρού χρησιμοποιούμε τη `fixUp` για να ανεβούμε στο σωρό (αν αυξάνεται η προτεραιότητα) ή τη `fixDown` για να κατεβούμε στο σωρό (αν μειώνεται η προτεραιότητα). Οι πλήρεις υλοποιήσεις τέτοιων λειτουργιών, οι οποίες αφορούν συγκεκριμένα στοιχεία δεδομένων, έχουν νόημα μόνο εφόσον διατηρείται μια αναφορά για κάθε στοιχείο, προς τη θέση αυτού του στοιχείου μέσα στη δομή δεδομένων. Θα εξετάσουμε αναλυτικά τις υλοποιήσεις με τις οποίες μπορούμε να το κάνουμε αυτό στις Ενότητες 9.5 έως 9.7.

Ιδιότητα 9.3 *Οι λειτουργίες* **αλλαγής προτεραιότητας**, **διαγραφής**, *και* **αντικατάστασης του μέγιστου** *για τον αφηρημένο τύπο δεδομένων ουράς προτεραιότητας μπορούν να υλοποιηθούν με δένδρα διατεταγμένα σε σωρό, έτσι ώστε να μην απαιτούνται περισσότερες από* $2 \lg N$ *συγκρίσεις για οποιαδήποτε λειτουργία σε μια ουρά με N στοιχεία.*



Εικόνα 9.5
Αναλυτική κατασκευή σωρού

Σε αυτή την ακολουθία παρουσιάζεται η εισαγωγή των κλειδιών A S O R T I N G *σε έναν αρχικά άδειο σωρό. Τα νέα στοιχεία προστίθενται στο κατώτατο σημείο του σωρού, μετακινούμενα από αριστερά προς τα δεξιά στο κατώτατο επίπεδο. Επειδή κάθε εισαγωγή επηρεάζει μόνο τους κόμβους της διαδρομής μεταξύ του σημείου εισαγωγής και της ρίζας, το κόστος χειρότερης περίπτωσης είναι ανάλογο του λογαρίθμου του μεγέθους του σωρού.*



Εικόνα 9.6
Αναλυτική κατασκευή σωρού (συνέχεια)

Σε αυτή την ακολουθία παρουσιάζεται η εισαγωγή των κλειδιών E X A M P L E *στο σωρό που αρχίσαμε να δημιουργούμε στην Εικόνα 9.5. Το συνολικό κόστος για την κατασκευή ενός σωρού μεγέθους N είναι λιγότερο από $1 + \lg 2 + \ldots + \lg N$ το οποίο είναι μικρότερο του $N \lg N$.*

Επειδή απαιτούν χειριστήρια για τα στοιχεία, θα αναβάλουμε την εξέταση των υλοποιήσεων που υποστηρίζουν αυτές τις λειτουργίες για την Ενότητα 9.6 (δείτε το Πρόγραμμα 9.12 και την Εικόνα 9.14). Όλες οι υλοποιήσεις περιλαμβάνουν μετακίνηση κατά μήκος μιας διαδρομής μέσα στο σωρό — ίσως ακόμη και ολόκληρη τη διαδρομή από επάνω προς τα κάτω ή από κάτω προς τα επάνω στη χειρότερη περίπτωση. ■

Παρατηρήστε ότι σε αυτή τη λίστα δεν περιλαμβάνεται η λειτουργία *ένωσης* (join). Ο αποδοτικός συνδυασμός δύο ουρών προτεραιότητας φαίνεται ότι απαιτεί μια πολύ πιο εξελιγμένη δομή δεδομένων. Θα εξετάσουμε αναλυτικά μια τέτοια δομή δεδομένων στην Ενότητα 9.7. Κατά τα άλλα, η απλή μέθοδος που βασίζεται σε σωρό και παρουσιάζεται εδώ είναι επαρκής για μια μεγάλη ποικιλία εφαρμογών. Χρησιμοποιεί ελάχιστο πρόσθετο χώρο και παρέχει εγγυήσεις αποδοτικής εκτέλεσης *εκτός* από την περίπτωση της παρουσίας συχνών και μεγάλων λειτουργιών *ένωσης*.

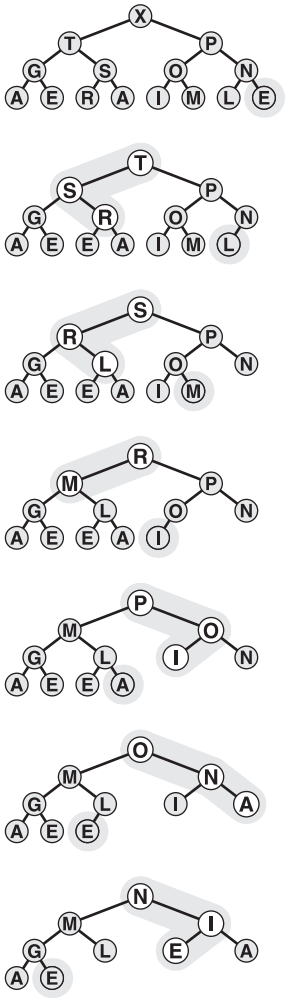
Όπως αναφέρθηκε, για να αναπτύξουμε μια μέθοδο ταξινόμησης μπορούμε να χρησιμοποιήσουμε οποιαδήποτε ουρά προτεραιότητας, όπως φαίνεται στο Πρόγραμμα 9.6. Εισάγουμε όλα τα προς ταξινόμηση κλειδιά στην ουρά προτεραιότητας και στη συνέχεια χρησιμοποιούμε αναδρομικά τη λειτουργία *διαγραφής του μέγιστου* προκειμένου να αφαιρέσουμε όλα τα στοιχεία του σωρού με φθίνουσα σειρά. Η χρήση, με αυτόν τον τρόπο, μιας ουράς προτεραιότητας η οποία αναπαρίσταται με μη διατεταγμένη λίστα αντιστοιχεί στην ταξινόμηση με επιλογή· αντίθετα, η χρήση διατεταγμένης λίστας αντιστοιχεί στην ταξινόμηση με εισαγωγή.

Στις Εικόνες 9.5 και 9.6 παρουσιάστηκε ένα παράδειγμα της πρώτης φάσης (της διαδικασίας κατασκευής) όταν χρησιμοποιείται μια υλοποίηση ουράς προτεραιότητας βασισμένη σε σωρό. Στις Εικόνες 9.7 και 9.8 παρουσιάζεται η δεύτερη φάση (η οποία αποκαλείται διαδικασία *ταξινόμησης προς τα κάτω*, sortdown) για την υλοποίηση που βασίζεται σε σωρό. Για πρακτικούς λόγους, αυτή η μέθοδος είναι συγκριτικά άκομψη, επειδή δημιουργεί ένα πρόσθετο αντίγραφο των προς ταξινόμηση στοιχείων (στην ουρά προτεραιότητας) χωρίς να είναι απαραίτητο. Επίσης, η χρήση N διαδοχικών εισαγωγών δεν είναι ο αποδοτικότερος τρόπος κατασκευής ενός σωρού από N δεδομένα στοιχεία. Στην επόμενη ενότητα θα ασχοληθούμε με αυτά τα δύο σημεία κατά την εξέταση μιας υλοποίησης του κλασικού αλγορίθμου heapsort.

Πρόγραμμα 9.6 Ταξινόμηση με ουρά προτεραιότητας

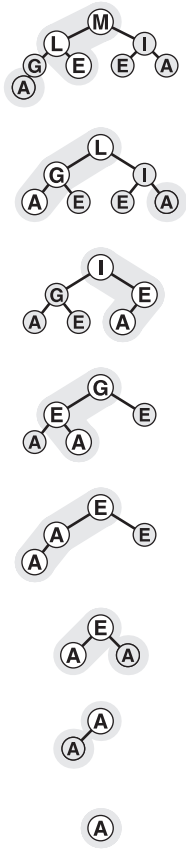
Για να ταξινομήσουμε έναν υποπίνακα `a[l], …, a[r]` χρησιμοποιώντας έναν ΑΤΔ ουράς προτεραιότητας, απλώς χρησιμοποιούμε τη συνάρτηση `PQinsert` για να τοποθετήσουμε όλα τα στοιχεία στην ουρά προτεραιότητας, και έπειτα χρησιμοποιούμε την `PQdelmax` για να τα αφαιρέσουμε, σε φθίνουσα σειρά. Αυτός ο αλγόριθμος ταξινόμησης εκτελείται σε χρόνο ανάλογο του $N \lg N$ αλλά χρησιμοποιεί πρόσθετο χώρο που είναι ανάλογος του πλήθους των προς ταξινόμηση στοιχείων (για την ουρά προτεραιότητας).

```
void PQsort(Item a[], int l, int r)
  { int k;
    PQinit();
    for (k = l; k <= r; k++) PQinsert(a[k]);
    for (k = r; k >= l; k--) a[k] = PQdelmax();
  }
```



Εικόνα 9.7
Ταξινόμηση από σωρό

Αφού αντικαταστήσουμε το μεγαλύτερο στοιχείο ενός σωρού με αυτό που βρίσκεται στο δεξιό άκρο του κατώτερου επιπέδου, μπορούμε να αποκαταστήσουμε τη διάταξη του σωρού μετακινούμενοι κατά μήκος μιας διαδρομής από τη ρίζα προς τα κάτω.



Εικόνα 9.8
Ταξινόμηση από σωρό (συνέχεια)

Σε αυτή την ακολουθία παρουσιάζεται η διαγραφή των υπόλοιπων κλειδιών του σωρού της Εικόνας 9.7. Ακόμη και αν κάθε στοιχείο μετακινηθεί κατά μήκος ολόκληρης της διαδρομής μέχρι το κατώτατο επίπεδο, το συνολικό κόστος της φάσης ταξινόμησης είναι λιγότερο από $\lg N + \ldots + \lg 2 + \lg 1$*, το οποίο είναι μικρότερο του* $N \log N$*.*