Δομές Δεδομένων και Αλγόριθμοι

Εισαγωγή

Χρήστος Γκόγκος

Πανεπιστήμιο Ιωαννίνων, Τμήμα Πληροφορικής και Τηλεπικοινωνιών (2019-2020)

Τι είναι αλγόριθμος;

> *Αλγόριθμος είναι μια σειρά από βήματα που επιτυγχάνουν έναν σκοπό. Οι* ***αλγόριθμοι υπολογιστών*** *πρέπει να περιγράφονται σε τέτοιο βαθμό λεπτομέρειας που να επιτρέπει την εκτέλεσή τους σε υπολογιστές.*

Κάθε αλγόριθμος υπολογιστή θα πρέπει:

- να παράγει πάντα ορθές λύσεις για οποιαδήποτε είσοδο δέχεται.
- να χρησιμοποιεί με αποδοτικό τρόπο τους υπολογιστικούς πόρους.

Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα (Niklaus Wirth)

Γιατί να μελετήσει κανείς αλγόριθμους και δομές δεδομένων;

- Προκειμένου να είναι σε θέση να αναπτύσσει αποδοτικές λύσεις σε προβλήματα που προκύπτουν κατά την ανάπτυξη υπολογιστικών εφαρμογών.
- Να αντιλαμβάνεται την ορολογία που χρησιμοποιείται από τους επαγγελματίες πληροφορικής.
- Να είναι σε θέση να εκτιμήσει το υπολογιστικό κόστος αλγοριθμικών λύσεων καθώς και την **εφικτότητα** ορισμένων από αυτές.

Δεν υπάρχει μια γενική διαδικασία για τη δημιουργία αλγορίθμων. Πολλές φορές η εύρεση του κατάλληλου αλγορίθμου απαιτεί πρωτότυπη σκέψη. Ωστόσο, υπάρχουν γενικές κατηγορίες μεθόδων που μπορούν να εφαρμοστούν με λιγότερο ή περισσότερο καλά αποτελέσματα (π.χ. διαίρει και βασίλευε, άπληστη μέθοδος, δυναμικός προγραμματισμός κλπ).

Μερικοί "διάσημοι" αλγόριθμοι

- Αλγόριθμος του Ευκλείδη για εύρεση του Μέγιστου Κοινού Διαιρέτη δύο ακεραίων αριθμών
- Quicksort, γρήγορη ταξινόμηση
- Αλγόριθμος του Dijkstra για εύρεση της συντομότερης διαδρομής από μια κορυφή προς όλες τις άλλες κορυφές ενός γραφήματος
- Αλγόριθμος Simplex (Linear Programming) για την επίλυση προβλημάτων Γραμμικού Προγραμματισμού
- Αλγόριθμοι συμπίεσης δεδομένων (jpeg, mp3, mmpeg-4)
- Αλγόριθμος Diffie-Hellman ανταλλαγής κλειδιών (κρυπτογραφία)
- Αλγόριθμος Pagerank (αλγόριθμος κατάταξης ιστοσελίδων της Google)

Παράδειγμα: Εύρεση ζεύγους τιμών με το μεγαλύτερο γινόμενο από ένα σύνολο τιμών

Δίνεται μια ακολουθία από μη αρνητικούς ακεραίους στο διάστημα [0,100000) και ζητείται να βρεθεί το ζεύγος τιμών με το μεγαλύτερο γινόμενο.

- Αν υπάρχουν περισσότεροι από ένας αλγόριθμοι που λύνουν **ορθά** το πρόβλημα, τότε θα πρέπει να επιλεχθεί ο καλύτερος.
- Τι σημαίνει καλύτερος αλγόριθμος;
  - σε σχέση με το χρόνο εκτέλεσης;
  - σε σχέση με τις απαιτήσεις του σε μνήμη;
  - σε σχέση με την ευκολία γραφής του ίδιου του αλγορίθμου;

Δημιουργία 1000 ακεραίων τυχαίων τιμών στο διάστημα [0,100000)

```
#include <random>
#include <ctime>
#define N 1000
#define UL 100000
int main()
{
    srand(time(NULL));
    int a[N];
    for (int i = 0; i < N; i++)
        a[i] = rand() % UL;
    ...
}
```

Ενδέχεται να προκύψει ***πρόβλημα*** *με τη δημιουργία μεγάλων τυχαίων αριθμών καθώς η rand() δημιουργεί ψευδοτυχαίες ακέραιες τιμές στο διάστημα [0, RAND_MAX]. Σε ορισμένες εγκαταστάσεις (π.χ. Windows, mingw-64 8.1) το RAND_MAX είναι μόνο 32767.*

Υπερχείλιση ακεραίων

Προσοχή σε περιπτώσεις υπερχείλισης ακεραίων.

```
#include <iostream>
#include <climits>

int main()
{
    std::cout << INT_MAX << std::endl;
    std::cout << LONG_MAX << std::endl;
    std::cout << LONG_LONG_MAX << std::endl;
}
```

όρια ακεραίων στη C++ (ενδέχεται να επιστραφούν διαφορετικά αποτελέσματα ανάλογα με το μεταγλωττιστή και το λειτουργικό σύστημα)

```
MAX INT      :             2,147,483,647
LONG INT     : 9,223,372,036,854,775,807
LONG LONG INT: 9,223,372,036,854,775,807
```

*Στο παράδειγμα, εάν δύο τυχαίες τιμές λάβουν την τιμή 99,999 (ή κάποια άλλη επαρκώς μεγάλη τιμή) τότε 99,999 * 99,999 = 9,999,800,001 > 2,147,483,647 άρα θα συμβεί υπερχείλιση.*

Παράδειγμα, Λύση 1 (με long long)

Πριν πολλαπλασιαστούν οι τιμές a[i] και a[j] γίνονται cast σε long long έτσι ώστε να αποφευχθεί πιθανή υπερχείλιση.

```
long long max = 0;
for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
        if ((long long)a[i] * (long long)a[j] > max)
            max = (long long)a[i] * (long long)a[j];
```

Παράδειγμα, Λύση 2

Εύρεση των δύο μεγαλύτερων τιμών της ακολουθίας και υπολογισμός του γινομένου τους.

```
int max1 = a[0], max2 = a[1];
if (max1 < max2)
{
    max1 = a[1];
    max2 = a[0];
}
for (int i = 2; i < N; i++)
    if (a[i] > max1)
    {
        max2 = max1;
        max1 = a[i];
    }
    else if (a[i] > max2)
        max2 = a[i];
std::cout   << "Maximum product is "
            << (long long)max1 * (long long)max2
            << std::endl;
```

Παράδειγμα, Λύση 3

Ταξινόμηση των τιμών της ακολουθίας σε αύξουσα σειρά και πολλαπλασιασμός των δύο τελευταίων στοιχείων της ακολουθίας. Είναι καλύτερος ο συγκεκριμένος αλγόριθμος από τον προηγούμενο;

```
#include <algorithm>
...
std::sort(std::begin(a), std::end(a));
std::cout   << "Maximum product is "
            << (long long)a[N - 1] * (long long)a[N - 2]
            << std::endl;
```

Εμπειρική εκτίμηση του χρόνου εκτέλεσης των 3 αλγορίθμων για N=100.000

```
Maximum product is 9999900000
#1 Time: 7733340micros = 7.73334s
Maximum product is 9999900000
#2 Time: 998micros = 0.000998s
Maximum product is 9999900000
#3 Time: 12965micros = 0.012965s

Maximum product is 9999600003
#1 Time: 7728326micros = 7.72833s
Maximum product is 9999600003
#2 Time: 0micros = 0s
Maximum product is 9999600003
#3 Time: 12965micros = 0.012965s
```

Ενεργοποίηση της βελτιστοποίησης κατά την παραγωγή του εκτελέσιμου από το μεταγλωττιστή (g++)

- Κλήση του μεταγλωττιστή με την παράμετρο -O2

```
$ g++ code.cpp -O2
$ ./a.out
Maximum product is 9999700002
#1 Time: 2731693micros = 2.73169s
Maximum product is 9999700002
#2 Time: 0micros = 0s
Maximum product is 9999700002
#3 Time: 4985micros = 0.004985s
```

- Μείωση του χρόνου εκτέλεσης για την περίπτωση #1 από 7.73 seconds σε 2.73 seconds.

Μέτρηση απόδοσης αλγορίθμων

- Η χρονομέτρηση της εκτέλεσης κώδικα δίνει μερική εικόνα για την αποδοτικότητα ενός αλγορίθμου. Εξαρτάται από:
  - Το σύνολο δεδομένων που δίνεται ως είσοδος.
  - Ταχύτητα υπολογιστή (χρονισμό επεξεργαστή, αρχιτεκτονική, ιεραρχίες μνήμης κ.α.).
  - Άλλα προγράμματα τα οποία εκτελούνται ταυτόχρονα στον υπολογιστή.
  - Τη γλώσσα προγραμματισμού στην οποία έγινε η υλοποίηση.
  - Τον μεταγλωττιστή και τις επιμέρους ρυθμίσεις λειτουργίας του.
  - Την επιδεξιότητα του προγραμματιστή
- Επίσης, για να χρονομετρηθεί ένας αλγόριθμος θα πρέπει **πρώτα** να γραφεί ο κώδικας σε κάποια γλώσσα προγραμματισμού.
- Ένας άλλος τρόπος εκτίμησης της απόδοσης ενός αλγορίθμου είναι μέσω της θεωρητικής ανάλυσής του και του προσδιορισμού της **Ασυμπωτικής Πολυπλοκότητας** του αλγορίθμου.