Δομές Δεδομένων και Αλγόριθμοι

Αναδρομή (V1.0)

Χρήστος Γκόγκος

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

Αναδρομή (1/2)

- Ένας αλγόριθμος λέγεται ότι είναι αναδρομικός αν καλεί τον εαυτό του.
- Κάθε αναδρομικός αλγόριθμος θα πρέπει:
  - Να χειρίζεται μια βασική περίπτωση χωρίς αναδρομή (base case).
  - Με κάθε αναδρομική κλήση να χειρίζεται ένα τμήμα του συνολικού προβλήματος (recursive case).
  - Να αποφεύγει ατέρμονους κύκλους κλήσεων (συνήθως αυτό σημαίνει ότι σταδιακά ασχολούμαστε με προβλήματα μικρότερης διάστασης).

```
void f(int n)
{
    if (n == 0) return;
    f(n - 1);
    cout << n << endl;
}

int main()
{
    f(5);
}
```

*Ο κώδικας θα εμφανίσει **1,2,3,4,5**.*

Αναδρομή (2/2)

- Αλλαγή στη θέση της αναδρομικής κλήσης της συνάρτησης οδηγεί σε αλλαγή της συμπεριφοράς του προγράμματος.

```
void f(int n)
{
    if (n == 0) return;
    cout << n << endl;
    f(n - 1);
}

int main()
{
    f(5);
}
```

*Ο κώδικας θα εμφανίσει **5,4,3,2,1**.*

Γιατί να χρησιμοποιήσει κανείς αναδρομή;

- Ο αναδρομικός κώδικας είναι συνήθως συντομότερος και ευκολότερος στη συγγραφή από ότι ο επαναληπτικός κώδικας.
- Η αναδρομή είναι ιδιαίτερα χρήσιμη σε εργασίες που ορίζονται με όρους παρόμοιων υποεργασιών (π.χ. η ταξινόμηση, η αναζήτηση και τα προβλήματα διάσχισης έχουν διαισθητικά απλές αναδρομικές λύσεις).

Παραγοντικό (1/2)

Το παραγοντικό (factorial) ορίζεται αναδρομικά για μη αρνητικούς ακέραιους αριθμούς $n \geq 0$ ως εξής:

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

```
int factorial(int n) {
    if (n == 0) return 1;
    else return n * factorial(n - 1);
}
int main() {
    cout << factorial(4) << endl;
}
```

Κατά την εκτέλεση ο κώδικας θα εμφανίσει την τιμή 4!=24.

Παραγοντικό (2/2)

Επαναληπτική έκδοση υπολογισμού του παραγοντικού.

```
int factorial(int n) {
    int f = 1;
    for (int i = 2; i <= n; i++)
        f = f * i;
    return f;
}
int main() {
    cout << factorial(4) << endl;
}
```

Κατά την εκτέλεση ο κώδικας θα εμφανίσει την τιμή 4!=24.

Αναδρομή ή επανάληψη (1/2)

- Αναδρομή
  1. Τερματίζει όταν φθάσει στη βασική περίπτωση.
  2. Κάθε αναδρομική κλήση απαιτεί επιπλέον χώρο στη μνήμη.
  3. Μια ατέρμονη αναδρομή θα οδηγήσει σε εξάντληση μνήμης (stack overflow).
  4. Για συγκεκριμένα προβλήματα η αναδρομική λύση είναι διαισθητικά απλούστερη.
- Επανάληψη
  1. Τερματίζει όταν η συνθήκη επανάληψης γίνει ψευδής.
  2. Κάθε επανάληψη πραγματοποιείται χωρίς επιπλέον απαιτήσεις μνήμης.
  3. Μια ατέρμονη επανάληψη δεν θα επιτρέψει στον κώδικα να τερματίσει αλλά δεν θα απαιτείται επιπλέον μνήμη σε κάθε επανάληψη.
  4. Η επαναληπτική λύση σε ορισμένα προβλήματα δεν είναι τόσο εύκολο να συλληφθεί προγραμματιστικά όσο η αναδρομική λύση.

Αναδρομή ή επανάληψη (2/2)

- Γενικά, οι αναδρομικές λύσεις είναι λιγότερο αποδοτικές από τις επαναληπτικές λύσεις (η επιβάρυνση στις αναδρομικές κλήσεις οφείλεται στις διαδοχικές κλήσεις συναρτήσεων).
- Με τη χρήση στοίβας μπορεί να πραγματοποιηθεί η μετατροπή οποιουδήποτε αναδρομικού αλγορίθμου σε επαναληπτικό (αυτό όμως δεν σημαίνει ότι αξίζει να γίνει κάτι τέτοιο).

Μέγιστος Κοινός Διαιρέτης (1/3)

- Ο Μέγιστος Κοινός Διαιρέτης (ΜΚΔ) 2 μη αρνητικών ακεραίων a και b είναι ο μεγαλύτερος ακέραιος που διαιρεί ακριβώς και τους δύο αριθμούς.
- Ο ΜΚΔ χρησιμοποιείται στην απλοποίηση κλασμάτων (για παράδειγμα γνωρίζοντας ότι οι ακέραιοι 357 και 234 έχουν ΜΚΔ 3, μπορούμε να απλοποιήσουμε το μεταξύ τους κλάσμα $\frac{357}{234}$ διαιρώντας αριθμητή και παρονομαστή με το ΜΚΔ, οπότε προκύπτει $\frac{\frac{357}{3}}{\frac{234}{3}} = \frac{119}{78}$).
- Υπάρχουν πολλές χρήσεις του ΜΚΔ στην πράξη (π.χ. κρυπτογραφία)
- Υπάρχει αποδοτικός αλγόριθμος (ο αλγόριθμος του Ευκλείδη) για τον υπολογισμό του ΜΚΔ που είναι από τη φύση του αναδρομικός.

Μέγιστος Κοινός Διαιρέτης (2/3)

Λήμμα: Έστω r το υπόλοιπο της διαίρεσης του a με το b. Τότε ισχύει gcd(a,b)=gcd(r,b)=gcd(b,r)

Απόδειξη:

- a = b*q+r για κάποιο q
- Μια τιμή d διαιρεί ακριβώς το a και το b αν διαιρεί ακριβώς το b και το r, άρα ο κοινοί διαιρέτες των a και b είναι οι ίδιοι με τους κοινούς διαιρέτες των b και r

Μέγιστος Κοινός Διαιρέτης (3/3)

Αναδρομική λύση

```
int gcd(int a, int b) {
    if (b == 0)
        return a;
    int r = a % b;
    return gcd(b, r);
}
```

Μη αναδρομική λύση

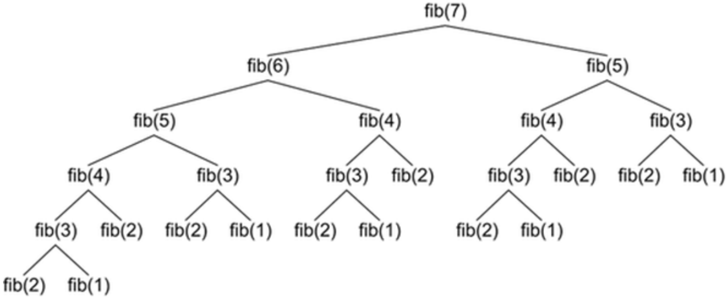
```
int gcd(int a, int b)
{
    while (b != 0){
        int r = a % b;
        a=b;
        b=r;
    }
    return a;
}
```

gcd(357,234) $\Rightarrow$ *gcd(234, 123)* $\Rightarrow$ *gcd(123, 111)* $\Rightarrow$ *gcd(111, 12)* $\Rightarrow$ *gcd(12, 3)* $\Rightarrow$ *gcd(3, 0)* $\Rightarrow$ *ο ΜΚΔ είναι το 3.*

Αριθμοί Fibonacci (1/2)

Υπολογισμός του n-οστού αριθμού Fibonacci με αναδρομή.

```
int fibo(int n)
{
    if (n <= 1)
        return n;
    else
        return fibo(n - 2) + fibo(n - 1);
}
```



Εκθετική πολυπλοκότητα !!!

Αριθμοί Fibonacci - memoization (2/2)

Memoization: αποθήκευση τιμών σε έναν πίνακα έτσι ώστε να μη χρειαστεί να γίνει κλήση της αναδρομικής συνάρτησης για να επαναϋπολογιστούν.

```
...
#define N 100
const int NIL = -1; int lookup_table[N];
void init() {
    for(int i=0; i<N; i++)
        lookup_table[i] = NIL;
}
int fib_mem(int n) {
    if(lookup_table[n] == NIL) {
        if(n <= 1)
            lookup_table[n] = n;
        else
            lookup_table[n] = fib_mem(n-1) + fib_mem(n-2);
    }
    return lookup_table[n];
}
int main() {
    init(); cout<<fib_mem(7) << endl;
}
```

Αναδρομικές εξισώσεις

Αν υποθέσουμε ότι η διαίρεση ενός προβλήματος διάστασης n γίνεται σε α υποπροβλήματα, κάθε υποπρόβλημα έχει μέγεθος $\frac{1}{b}$ *του προβλήματος από το οποίο έχει προκύψει και ότι το βήμα συνδυασμού χρειάζεται* $d(n)$ *χρόνο τότε προκύπτει η ακόλουθη αναδρομική εξίσωση:*

$$T(n) = \begin{cases} 1 & \text{if} \quad n = 1 \\ aT(\frac{n}{b}) + d(n) & \text{if} \quad n > 1 \end{cases}$$

Αναδρομικές εξισώσεις - μέθοδος αντικατάστασης

$$
\begin{aligned}
T(n) &= aT(n/b) + d(n) \\
&= a[aT(n/b^2) + d(n/b)] + d(n) \\
&= a^2T(n/b^2) + ad(n/b) + d(n) \\
&= a^3T(n/b^3) + a^2d(n/b^2) + ad(n/b) + d(n) \\
&\dots \\
&= a^kT(n/b^k) + \sum_{i=0}^{k-1} a^i d(n/b^i)
\end{aligned}
$$

Θεωρώντας ότι $n = b^k$ και με δεδομένες κατά περίπτωση τις τιμές των a και b μπορούμε να οδηγηθούμε σε απλοποιημένη μορφή για το $T(n)$ με αντικαταστάσεις. Για παράδειγμα εάν έχουμε $a = 2$, $b = 2$ και $d(n) = n - 1$, τότε σταδιακά προκύπτει:

$$
\begin{aligned}
T(n) &= n + \sum_{i=0}^{k-1} 2^i(2^{k-i} - 1) \\
&= n + \sum_{i=0}^{k-1} (2^k - 2^i) \\
&= n + 2^k k - \frac{2^k - 1}{2 - 1} \\
&= n + n\log n - n + 1 \\
&= n\log n + 1
\end{aligned}
$$

Το master θεώρημα

Έστω μια αναδρομική εξίσωση της ακόλουθης μορφής:

$$T(n) = aT(\frac{n}{b}) + d(n) \quad d(n) = \mathrm{O}(n^k)$$

για την οποία γνωρίζουμε ότι:

- $a =$ αριθμός αναδρομικών κλήσεων ($a \geq 1$)
- $b =$ συντελεστής συρρίκνωσης του προβλήματος ($b > 1$)
- $k =$ εκθέτης του n για το χρόνο εκτέλεσης που απαιτεί το μη αναδρομικό τμήμα του αλγορίθμου

τότε ισχύει:

- $T(n) = \mathrm{O}(n^k \log n)$ αν $a = b^k$
- $T(n) = \mathrm{O}(n^k)$ αν $a < b^k$
- $T(n) = \mathrm{O}(n^{\log_b a})$ αν $a > b^k$

Εφαρμογές του master θεωρήματος (1/2)

Ποια είναι η ασυμπτωτική πολυπλοκότητα που προκύπτει για την αναδρομική σχέση: $T(n) = 2T(\frac{n}{2}) + \mathrm{O}(n)$;

- Προκύπτει από την αναδρομική σχέση ότι $a = 2, b = 2, k = 1$. Άρα $a = b^k$ καθώς $2 = 2^1$ και συνεπώς η ασυμπτωτική πολυπλοκότητα είναι $\mathrm{O}(n \log n)$.

Ποια είναι η ασυμπτωτική πολυπλοκότητα που προκύπτει για την αναδρομική σχέση: $T(n) = T(\frac{n}{2}) + \mathrm{O}(1)$;

- Προκύπτει από την αναδρομική σχέση ότι $a = 1, b = 2, k = 0$. Άρα $a = b^k$ καθώς $1 = 2^0$ και συνεπώς η ασυμπτωτική πολυπλοκότητα είναι $\mathrm{O}(\log n)$.

Εφαρμογές του master θεωρήματος (2/2)

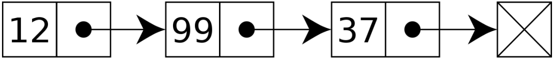
Ποια είναι η ασυμπτωτική πολυπλοκότητα που προκύπτει για την αναδρομική σχέση: $T(n) = 3T(\frac{n}{2}) + \mathrm{O}(n)$;

- Προκύπτει από την αναδρομική σχέση ότι $a = 3, b = 2, k = 1$. Άρα $a > b^k$ καθώς $3 > 2^1$ και συνεπώς η ασυμπτωτική πολυπλοκότητα είναι $\mathrm{O}(n^{\log_2 3}) = \mathrm{O}(n^{1.58})$.

Ποια είναι η ασυμπτωτική πολυπλοκότητα που προκύπτει για την αναδρομική σχέση: $T(n) = 3T(\frac{n}{3}) + \mathrm{O}(n^2)$;

- Προκύπτει από την αναδρομική σχέση ότι $a = 3, b = 3, k = 2$. Άρα $a < b^k$ καθώς $3 < 2^2 = 4$ και συνεπώς η ασυμπτωτική πολυπλοκότητα είναι $\mathrm{O}(n^2)$.

Αναδρομικές δομές (συνδεδεμένη λίστα)




```
...
struct node {
    int value;
    node *next = NULL;
};
int length(node *head){
    node *current = head; int c = 0;
    while (current != NULL) {c++; current = current->next;}
    return c;
}
int main() {
    node n1, n2, n3;
    n1.value = 12; n1.next = &n2;
    n2.value = 99; n2.next = &n3;
    n3.value = 37; node *head = &n1;
    cout << "The length of the list is: " << length(head) << endl;
}
```