Δομές Δεδομένων

5η Διάλεξη

Λίστες και αρχές ανάλυσης αλγορίθμων

Ε. Μαρκάκης

Περίληψη

- Διπλά συνδεδεμένες λίστες
- Αναπαράσταση γράφων με λίστες
- Εμπειρική ανάλυση αλγορίθμων
- Μαθηματική ανάλυση αλγορίθμων
- Αύξηση συναρτήσεων
- Συμβολισμός μεγάλου όμικρον
- Βασικές αναδρομικές εξισώσεις
- Παραδείγματα ανάλυσης
- Όρια ανάλυσης αλγορίθμων

Διπλά συνδεδεμένη λίστα

- Διπλά συνδεδεμένη λίστα
  - 2 δείκτες σε κάθε κόμβο
  - Κάθε κόμβος δείχνει και στον προηγούμενο
  - Κίνηση προς δύο κατευθύνσεις
  - Ευκολότερη εισαγωγή και διαγραφή
    - Εισαγωγή: αρκεί να έχουμε δείκτη προς προηγούμενο ή επόμενο κόμβο
    - Διαγραφή: αρκεί ο προς διαγραφή κόμβος
  - Μεγαλύτερο κόστος συντήρησης
    - Διπλάσιοι σύνδεσμοι προς ενημέρωση
  - Μεγαλύτερο κόστος μνήμης
    - Διπλάσιοι σύνδεσμοι προς αποθήκευση

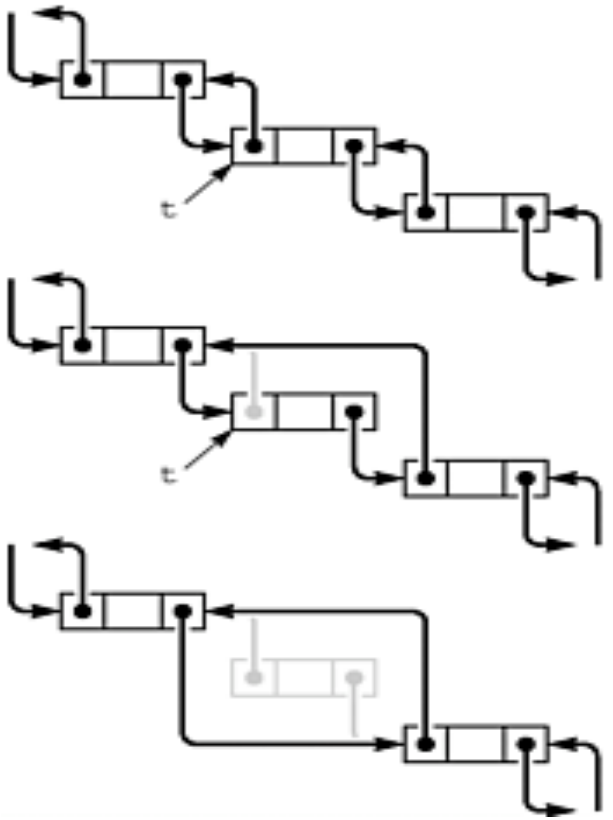
Διπλά συνδεδεμένη λίστα

- Δήλωση διπλά συνδεδεμένης λίστας

```
class Node{
  Object item;
  Node next;
  Node prev;
  Node(Object v){
       item = v;
       next = null;
       prev = null;
  } }
```

- Διαγραφή του κόμβου t
  - Δεν χρειάζονται άλλοι κόμβοι

```
t.next.prev = t.prev;
t.prev.next = t.next;
```



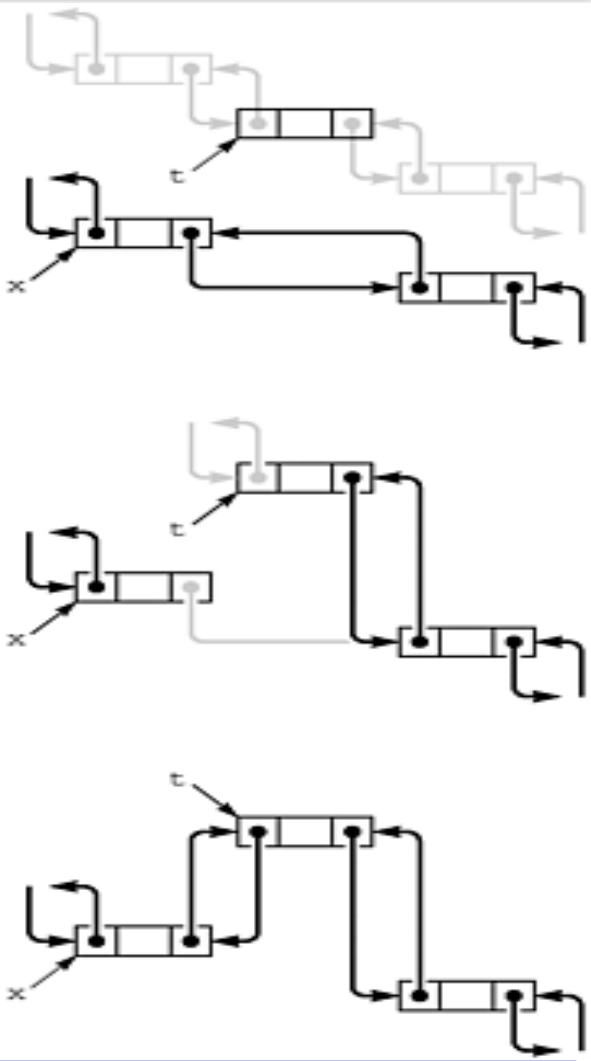
Διπλά συνδεδεμένη λίστα

- Εισαγωγή του κόμβου t
  - Αρκεί ο προηγούμενος ή ο επόμενος
  - Έστω ότι δίνεται ο προηγούμενος x

```
t.next = x.next;
x.next.prev = t;
x.next = t;
t.prev = x;
```

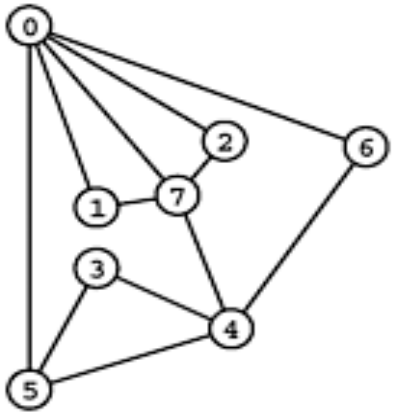
  - Έστω ότι δίνεται ο επόμενος y

```
t.prev = y.prev;
y.prev.next = t;
y.prev = t;
t.next = y;
```



Επιστροφή στην Αναπαράσταση Γράφων

- *Γράφος (graph)*: (V, E)
  - V: Ένα σύνολο από κόμβους
  - E: Πλευρές (σύνδεσμοι μεταξύ κόμβων)
  - Αν έχουμε N κόμβους, τους ονοματίζουμε από 0 ως N-1
- Πίνακας γειτνίασης
  - Συμμετρικός πίνακας NxN
  - Αν (i, j)∈E, `a[i][j] = a[j][i] = 1`
  - Διαφορετικά `a[i][j] = a[j][i] = 0`
  - Από σύμβαση θέτουμε `a[i][i] = 1` ∀i
  - Χώρος μνήμης: N^2
  - Χρόνος για να δούμε αν συνδέονται 2 κορυφές: σταθερός, ανεξάρτητος από N (1 εντολή)



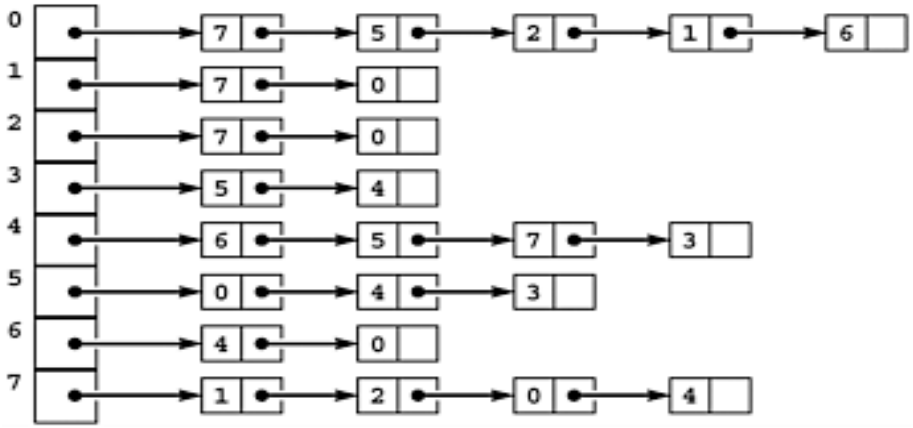
	0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1
2	1	0	1	0	0	0	0	1
3	0	0	1	1	1	0	0	0
4	0	0	0	1	1	1	1	0
5	1	0	0	1	1	1	0	0
6	1	0	0	0	1	0	1	0
7	1	1	1	0	1	0	0	1

Δημιουργία πίνακα γειτνίασης

```
class AdjacencyMatrix {
    public static void main(String[] args) {
    int N = Integer.parseInt(args[0]);
    int E = Integer.parseInt(args[1]);
    boolean adj[][] = new boolean[N][N];
    for (int i = 0; i < N; i++)
         for (int j = 0; j < N; j++)
             adj[i][j] = false;
    for (int i = 0; i < N; i++) adj[i][i] = true;
    for (In.init(); !In.empty() ;) {
         int i = In.getInt(), j = In.getInt();
         adj[i][j] = true; adj[j][i] = true;
    }
} }
```

Αναπαράσταση με λίστες γειτνίασης

- Διατηρούμε έναν πίνακα με λίστες
  - Μία λίστα για κάθε κορυφή
  - Περιέχει όλες τις άλλες κορυφές που συνδέονται μαζί της
  - Χώρος μνήμης: $N + |E|$ ($|E| << N^2$)
  - Για αραιούς γράφους πολύ προτιμότερη η χρήση λίστας αντί πίνακα
  - Χρόνος για να δούμε αν συνδέονται 2 κορυφές: πρέπει να διατρέξουμε τη λίστα της μίας εκ των 2 κορυφών, στη χειρότερη περίπτωση: N



Δημιουργία λιστών γειτνίασης

```
class AdjacencyLists {
    static class Node
       { int v; Node next;
          Node (int v, Node t)
             { this.v = v; next = t; }
        }
  public static void main(String[] args)
  { int N = Integer.parseInt(args[0]);
    int E = Integer.parseInt(args[1]);
    Node adj[] = new Node[N];
    for (int i = 0; i < N; i++) adj[i] = null;
    for (In.init(); !In.empty() ;) {
       int i = In.getInt(), j = In.getInt();
       adj[j] = new Node(i, adj[j]);
       adj[i] = new Node(j, adj[i]); } } }
```

Κεφάλαιο 2

Αρχές Ανάλυσης Αλγορίθμων

Εισαγωγή

- Γιατί αναλύουμε τους αλγορίθμους;
  - Πρόβλεψη επίδοσης: αναμενόμενος χρόνος εκτέλεσης
  - Σύγκριση αλγορίθμων: ποιος είναι ταχύτερος;
  - Παροχή εγγυήσεων: άνω όριο χρόνου εκτέλεσης
  - Κατανόηση θεωρητικής βάσης: συμπεριφορά του αλγορίθμου
- Επιστημονική ανάλυση και σύγκριση αλγορίθμων
  - Μπορεί να είναι είτε εμπειρική (υλοποίηση και σύγκριση του χρόνου εκτέλεσης)
  - Είτε θεωρητική (ανάλυση του αριθμού των απαιτούμενων βημάτων/εντολών ως συνάρτηση του μεγέθους των δεδομένων της εισόδου)

Εμπειρική ανάλυση

- Επιστημονική ανάλυση αλγορίθμων
  - Παρατήρηση ενός φαινομένου
  - Δημιουργία μία υπόθεσης / μοντέλου συνεπούς με το φαινόμενο
  - Πρόβλεψη γεγονότων με βάση το μοντέλο
  - Επαλήθευση των προβλέψεων μέσω παρατήρησης
  - Επικύρωση μέσω επανάληψης μέχρι να έχουμε συμφωνία
  - Τα πειράματα πρέπει να μπορούν να επαναληφθούν
- Υλοποίηση και εμπειρική ανάλυση
  - Κάθε εκτέλεση ενός προγράμματος είναι ένα πείραμα
  - Οι υλοποιήσεις μπορούν να συγκριθούν
    - Χρόνος εκτέλεσης, κατανάλωση μνήμης
  - Προσοχή: οι υλοποιήσεις δεν είναι όλες καλές!

Εμπειρική ανάλυση

- Στάδια εμπειρικής ανάλυσης
  1. Υλοποίηση και εκσφαλμάτωση του προγράμματος
     - Δεν θέλουμε κακές ή λανθασμένες υλοποιήσεις
     - Όλες οι υλοποιήσεις στο ίδιο περιβάλλον προγραμματισμού
  2. Προσδιορισμός της φύσης των δεδομένων
     - Πραγματικά: αυτά που εμφανίζονται στην πράξη
     - Τυχαία: τυχαία επιλογή από το πεδίο εισόδου (π.χ. με τη `Math.random()`)
     - Ακραία/μη ρεαλιστικά (perverse): επιλεγμένα έτσι ώστε να δυσκολεύουν τον αλγόριθμο
  3. Εκτέλεση της υλοποίησης με διάφορα μεγέθη προβλήματος

Εμπειρική ανάλυση

- Χρόνος εκτέλεσης υλοποίησης
  - Μπορεί να μετρηθεί χρησιμοποιώντας το χρονόμετρο
    - Κλήση μεθόδου System.currentTimeMillis()
  - Γενικά αυξάνεται όταν αυξάνεται το μέγεθος της εισόδου
  - Μπορεί να εξαρτάται αρκετά από τα δεδομένα εισόδου
    - Παράδειγμα: ταξινόμηση ενός σχεδόν ταξινομημένου πίνακα
  - Μπορεί να μας ενδιαφέρει ο μέσος ή ο μέγιστος χρόνος
  - Average case vs worst case analysis
- Από τι εξαρτάται ο χρόνος εκτέλεσης;
  - Μηχανή, μεταγλωττιστής, αλγόριθμος, δεδομένα εισόδου
  - Κρυφή μνήμη, συλλογή σκουπιδιών, φόρτος CPU
  - Δύσκολο να πάρουμε ακριβείς μετρήσεις με μία εκτέλεση
  - Συνήθως χρησιμοποιούμε μέσο όρο πολλών εκτελέσεων

Εμπειρική ανάλυση

- Πόσο σίγουρη είναι η εμπειρική ανάλυση;
  - Πιο σίγουρη σε παρόμοιες υλοποιήσεις
  - Παραλλαγές του ίδιου βασικού αλγόριθμου
  - Ίδια δεδομένα και περιβάλλον εκτέλεσης
- Η παγίδα των απλών αλγορίθμων
  - Ο απλός αλγόριθμος μπορεί να είναι πολύ αργός
  - Απαιτούμενη μνήμη μπορεί επίσης να είναι μεγάλη
- Η παγίδα της βελτιστοποίησης
  - Δεν είναι ανάγκη να βελτιστοποιούμε τα πάντα
  - Έχει σημασία ο συνολικός χρόνος εκτέλεσης
    - Αν το πρόγραμμα απαιτεί λίγα msec δεν έχει μεγάλα περιθώρια βελτίωσης
    - Το κόστος ανάπτυξης και συντήρησης μπορεί να είναι μεγάλο
- Βελτιστοποίηση εκεί που καθυστερεί το πρόγραμμα

Μαθηματική ανάλυση

- Χρόνος εκτέλεσης προγράμματος
  - Για κάθε εντολή: (χρόνος εκτέλεσης) * (συχνότητα εκτέλεσης)

```
for (int i = 0; i < n; i++)
    a[i] = 5;
```

  - Έστω ότι κάθε εντολή απαιτεί β μsec.
  - Η εντολή μέσα στο βρόχο εκτελείται n φορές, η εντολή `i++` n φορές και ο έλεγχος τερματισμού του βρόχου n+1 φορές
  - Συνολικός χρόνος: β(n+n+n+1 + 1) = β(3n+2)
  - Η παράμετρος β εξαρτάται από το σύστημα
  - Οι υπόλοιποι όροι εξαρτώνται γενικά από τον αλγόριθμο και τα δεδομένα
  - Στη μαθηματική ανάλυση αλγορίθμων δεν μας απασχολεί το β
  - Μας απασχολεί μόνο ο συνολικός αριθμός εντολών που πρέπει να εκτελεστούν (προσεγγιστικά)
  - Ανεξάρτητο από το σύστημα και τη γλώσσα προγραμματισμού
  - Σημαντικό: καθορισμός των πιο σημαντικών λειτουργιών

Μαθηματική ανάλυση

- Μειονεκτήματα μαθηματικής ανάλυσης
  - Θεωρητικά υπάρχουν ακριβή μαθηματικά μοντέλα
  - Στην πράξη οι τύποι μπορεί να είναι περίπλοκοι και δυσεπίλυτοι
  - Συνήθως καταφεύγουμε σε προσεγγιστικές αναλύσεις
- Πλεονεκτήματα μαθηματικής ανάλυσης
  - Δεν εξαρτάται από τα χαρακτηριστικά του περιβάλλοντος και της γλώσσας προγραμματισμού
  - Πρόβλεψη επίδοσης ως συνάρτηση του μεγέθους της εισόδου
  - Επιτρέπει πρόβλεψη επιδόσεων με πολύ μεγάλες εισόδους
  - Μηδενικό κόστος

Αύξηση συναρτήσεων

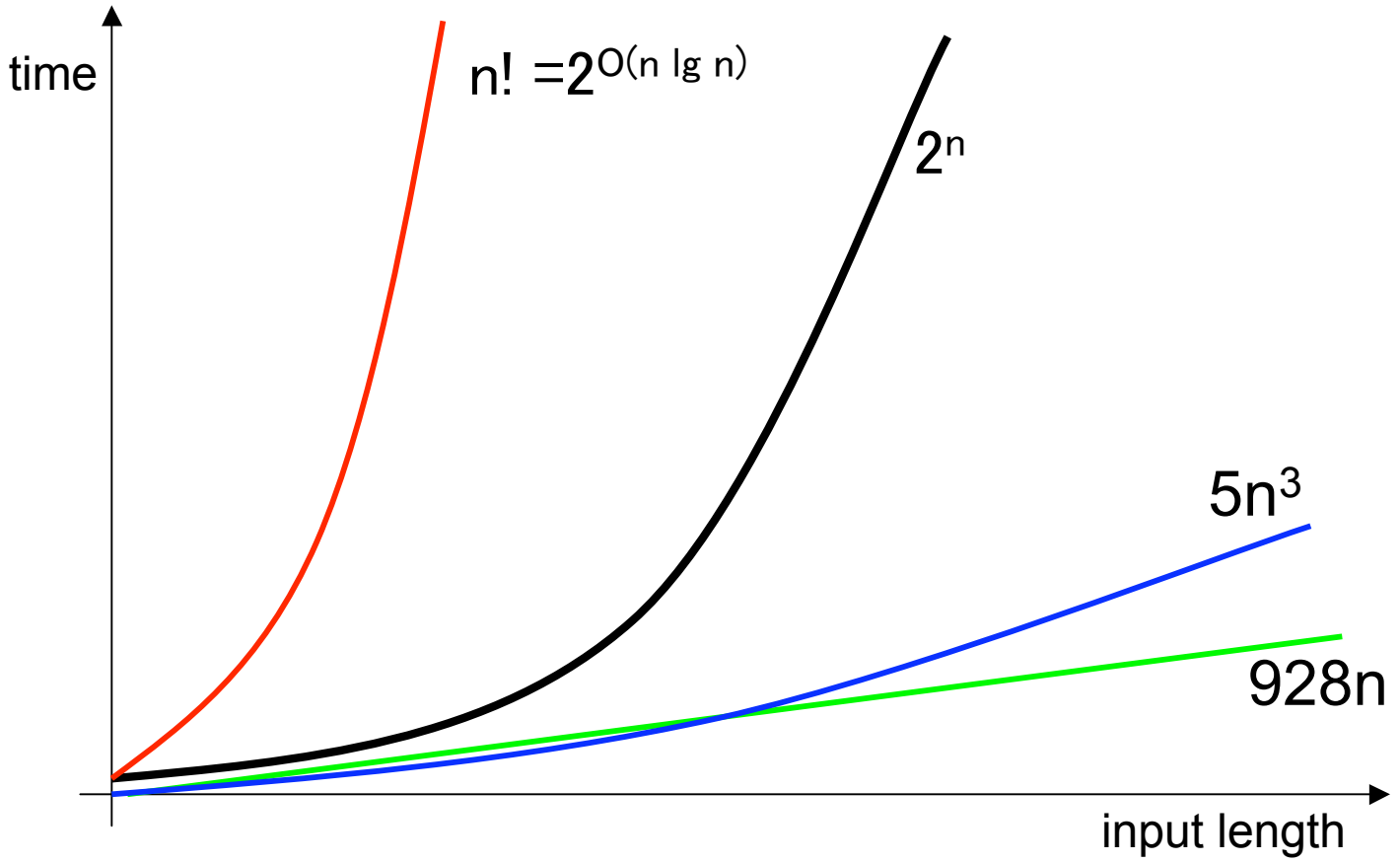
- Κύρια παράμετρος αλγορίθμου: N
  - Χαρακτηρίζει το μέγεθος του προβλήματος (π.χ. σε προβλήματα ταξιόμησης, είναι το πλήθος των αριθμών που ταξινομούμε)
- Έκφραση χρόνου εκτέλεσης σε συνάρτηση του N
  - **1:** σταθερός χρόνος εκτέλεσης, ανεξάρτητος του N.
  - **logN:** ελαφρά αύξηση καθώς αυξάνεται το N
    - Παράδειγμα: για 1000N ο χρόνος αυξάνεται κατά 3 (log10) ≈10
  - **N:** αύξηση ανάλογη της αύξησης του N.
  - **NlogN:** αύξηση λίγο πιο γρήγορη από αυτή του N.
  - **N^2:** τετραγωνική αύξηση του χρόνου, όχι και τόσο πρακτικός
    - Παράδειγμα: για 1000N ο χρόνος αυξάνεται κατά 10^6
  - **2^N:** Εκθετική αύξηση, μη πρακτικός ακόμα και για μικρά προβλήματα
    - Για N=20 ο χρόνος εκτέλεσης είναι > 1000000

Αύξηση συναρτήσεων

- Πρακτική μορφή συναρτήσεων του N στην ανάλυση προγραμμάτων
  - Συνήθως της μορφής a + c f(N)
  - Το c μπορεί να εκφράζει το πλήθος εντολών στον κύριο βρόχο
  - Το a εκφράζει όρους μικρότερης σημασίας
  - Το πιο σημαντικό είναι η μορφή της f(N)

logN	Sqrt(N)	N	NlogN	$N(\log N)^2$	$N^{3/2}$	N^2
7	10	100	664	4414	1000	10000
10	32	1000	9966	99317	31623	1000000
13	100	10000	132877	1765633	1000000	100000000
17	316	100000	1660964	27588016	31622777	10000000000
20	1000	1000000	19931569	397267426	1000000000	1000000000000

Αύξηση Συναρτήσεων



Επιπλέον Ορολογία

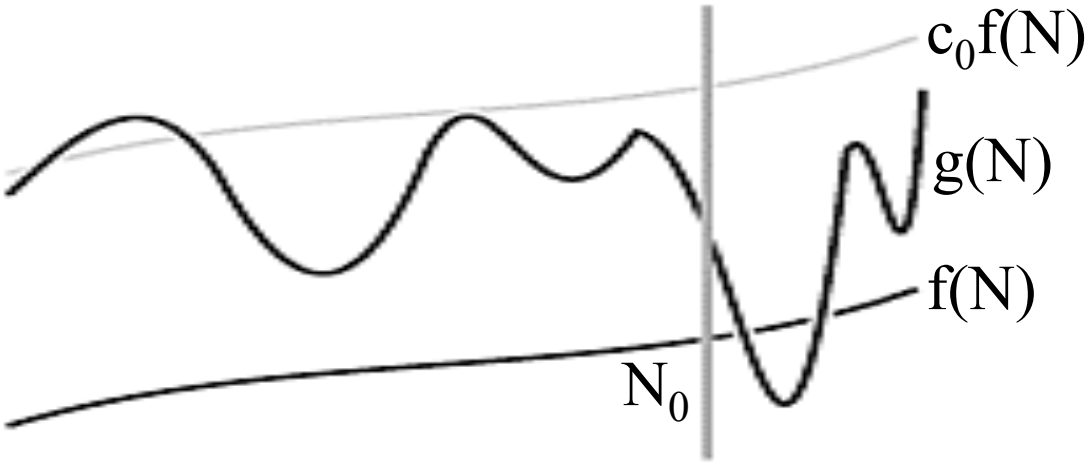
- $\lfloor x \rfloor$: ο μεγαλύτερος ακέραιος που είναι $\leq x$
  - $\lfloor 3.14 \rfloor = 3$, $\lfloor 9.99 \rfloor = 9$
- $\lceil x \rceil$: ο μικρότερος ακέραιος που είναι $\geq x$
  - $\lceil 3.14 \rceil = 4$, $\lceil 9.99 \rceil = 10$
  - $\lceil \log(N+1) \rceil$ = πλήθος των bits στη δυαδική αναπαράσταση του N
- $\log N = \log_2 N$, $\ln N = \log_e N$, για όλους τους άλλους λογαρίθμους, γράφουμε ρητά ποια είναι η βάση, π.χ. $\log_{10} N$
- H_N: N-οστός αρμονικός αριθμός, $H_N = 1+1/2+1/3+...+1/N \approx \ln N$

Συμβολισμός μεγάλου όμικρον

- Ορισμός συμβολισμού μεγάλου όμικρον
  - Λέμε ότι η συνάρτηση g(N) είναι O(f(N)) αν υπάρχουν θετικές σταθερές c_0 και N_0 έτσι ώστε $g(N) \leq c_0 f(N)$ για κάθε $N \geq N_0$
  - Παράδειγματα:
    - $2n + 10$ είναι O(n). Αρκεί να θέσουμε $c_0 = 3$ και $N_0 = 10$
    - $4N\log N + 150N + 3000\text{sqrt}(\log N) = O(N\log N)$. $c_0 = 3154$, $N_0 = 1$
  - Ερμηνεία: το $c_0 f(N)$ είναι άνω φράγμα της g(N) για μεγάλα N
  - Δεν ξέρουμε τι ισχύει για $N < N_0$ (και δεν μας ενδιαφέρει)
  - Το c_0 μπορεί να είναι πολύ μεγάλο
  - Προσοχή: g(N) είναι O(f(N)) δεν σημαίνει ότι f(N) είναι O(g(N))
- Χρήσεις συμβολισμού μεγάλου όμικρον
  - Άνω φράγμα σφάλματος όταν αγνοούμε μικρούς όρους
  - Άνω φράγμα σφάλματος όταν αγνοούμε κάποιες γραμμές κώδικα
  - Ταξινόμηση αλγορίθμων με βάση άνω φράγματα εκτέλεσης

Συμβολισμός μεγάλου όμικρον

- Διαγραμματική ερμηνεία g(N) = O(f(N))
  - g(N): συνάρτηση που μας ενδιαφέρει (π.χ. μπορεί να είναι ο χρόνος εκτέλεσης). Ακανόνιστη, ενδεχομένως με πολλούς ασήμαντους όρους
  - f(N): ομαλή συνάρτηση
  - O(f(N)) μας δείχνει την ασυμπτωτική συμπεριφορά της g(N)



Χειρισμός παραστάσεων με ασυμπτωτικούς όρους

- Αν το f(N) είναι πολυώνυμο βαθμού d, τότε είναι $O(N^d)$
  - Οι όροι με χαμηλότερο βαθμό απαλείφονται
  - Οι σταθεροί όροι απαλείφονται
- Χρησιμοποιούμε συνήθως το πιο αυστηρό όριο
  - Λέμε ότι 2N είναι O(N). Είναι και $O(N^2)$ αλλά O(N) είναι καλύτερο άνω φράγμα
- Οι σταθεροί πολλαπλασιαστές δεν έχουν σημασία
  - Λέμε ότι η 3N+5 είναι O(N). Είναι και O(3N) και O(N/2) αλλά ασυμπτωτικά δεν υπάρχει διαφορά μεταξύ O(N), O(3N), O(N/2)
- Σημασία έχει ο μεγαλύτερος ρυθμός αύξησης
  - Αν κάτι είναι N^2+O(N)+O(NlgN) τότε είναι $O(N^2)$
- Πράξεις σε παραστάσεις με ασυμπτωτικούς όρους: όπως με κανονικούς αριθμούς
    - $(N+O(1))(N+O(\log N) + O(1)) = N^2 + O(N\log N) + O(N)+ O(N) + O(\log N) + O(1) = N^2 + O(N\log N) + O(N) + O(\log N) + O(1) = O(N^2)$

Παραδείγματα

- $N^4 + 10N^3 + 80\ N^2 + 24 = O(N^4)$
- $2^N = O(2^{N+1})$ και $2^{N+1} = O(2^N)$
- $N^d = O(2^N)$ για κάθε σταθερά d.
- $2^N \neq O(N^d)$
- $(\log N)^m = O(N^d)$ για σταθερές m, d
- $O(f(N))O(g(N)) = O(f(N)g(N))$

Συμβολισμός μεγάλου όμικρον

- Πρόβλεψη χρόνου εκτέλεσης (T_N) με βάση το ρυθμό αύξησης
  - Πόσο αυξάνεται ο χρόνος εκτέλεσης όταν διπλασιάζεται το N;
  - O(1): $T_{2N} = T_N$
  - O(logN): $T_{2N} \sim T_N$ (λίγο παραπάνω)
  - O(N): $T_{2N} = 2T_N$
  - O(NlogN):$T_{2N} \sim 2T_N$ (λίγο παραπάνω)
  - $O(N^2)$: $T_{2N} \sim 4T_N$
  - $O(2^N)$: $T_{2N} \sim (T_N)^2$
- Μπορούμε να δουλέψουμε και αντίστροφα
  - Εκτελούμε το πρόγραμμα για είσοδο N και 2N
  - Διαιρούμε το χρόνο εκτέλεσης
  - Εκτιμούμε το ρυθμό αύξησης με βάση το αποτέλεσμα
  - Προσοχή: το N πρέπει να είναι αρκετά μεγάλο!