# Δομές Δεδομένων

3η Διάλεξη

Στοιχειώδεις Δομές Δεδομένων: Πίνακες

Ε. Μαρκάκης

# Αλλαγή αίθουσας

- Τις Παρασκευές 1-3, το μάθημα θα γίνεται στη Δ23

# Περίληψη

- Σύνοψη 1ου κεφαλαίου
- Δομικά στοιχεία
- Πίνακες
- Το κόσκινο του Ερατοσθένη
- Εύρεση πλησιέστερων σημείων
- Αντιγραφή πινάκων
- Αλλαγή μεγέθους πίνακα
- Πίνακες με 2 ή περισσότερες διαστάσεις
- Ειδικές κατηγορίες πινάκων

# Ανακεφαλαίωση των αλγορίθμων ένωσης - εύρεσης

- Τι κρατάμε από το προηγούμενο μάθημα:
  - Για κάθε πρόβλημα, αποσαφηνίζουμε πρώτα ποιες ακριβώς είναι οι απαιτήσεις (δεδομένα εισόδου, έξοδος, κτλ)
  - Σκεφτόμαστε ποιες είναι οι βασικές αφηρημένες λειτουργίες που πρέπει να υποστηρίζει το πρόγραμμα που θα φτιάξουμε (στο παράδειγμα: εύρεση, ένωση)
  - Φτιάχνουμε αρχικά μία απλή (έστω και μη αποδοτική, αργή) υλοποίηση (όπως η QuickFind)
  - Προσπαθούμε να βελτιώσουμε το χρόνο ή τη μνήμη βλέποντας πού υστερεί η αρχική μας υλοποίηση (stepwise refinement)

# Ανακεφαλαίωση των αλγορίθμων ένωσης - εύρεσης

- Τελικά συμπεράσματα για το παράδειγμα που είδαμε:
  - Για N αντικείμενα και M ζεύγη στην είσοδο, η σταθμισμένη γρήγορη ένωση με συμπίεση μονοπατιών τρέχει σε χρόνο «σχεδόν» ανάλογο του (M + N), ενώ η γρήγορη εύρεση χρειάζεται χρόνο ανάλογο του MN στη χειρότερη περίπτωση
  - Ο αλγόριθμος της σταθμισμένης γρήγορης ένωσης με συμπίεση μονοπατιών τρέχει πιο γρήγορα σε ένα κινητό που υποστηρίζει Java από ότι ο αλγόριθμος της γρήγορης εύρεσης σε ένα supercomputer

# Δομικά στοιχεία

- Δεδομένα
  - Τα πάντα αποσυντίθενται τελικά σε μεμονωμένα bits
  - Δύσκολο/επίπονο να επεξεργαζόμαστε bits
- Τύπος δεδομένων
  - Δομή / οργάνωση δεδομένων σε σύνολα από bits
  - Πράξεις / μέθοδοι πάνω στα δεδομένα
- Ενσωματωμένοι τύποι στη Java
  - `boolean`,
  - `char`,
  - `byte (8 bits), short (16 bits), int (32 bits), long (64 bits)`
  - `float (32 bits), double (64 bits)`
  - Οι μεταβλητές των τύπων αυτών περιέχουν δεδομένα

# Δομικά στοιχεία

- Κλάση ή Τάξη (Class) στη Java
  - Γενίκευση του τύπου δεδομένων
  - Δεδομένα και μέθοδοι
- Αντικείμενα στη Java
  - Οτιδήποτε κατασκευάζεται μέσα από μία τάξη
  - Οι μεταβλητές των τύπων αυτών περιέχουν αναφορές
- Διαφορές αντικειμένων από ενσωματωμένους τύπους
  - Τα αντικείμενα πρέπει να δημιουργηθούν με τον τελεστή new
  - Στα αντικείμενα δεν μπορούμε να ορίσουμε τελεστές
  - Υπάρχουν κλάσεις που ενθυλακώνουν τα αντικείμενα
    - `Boolean, Char, Byte, …`

# Δομικά στοιχεία

- Μετατροπές τύπων
  - Υπονοούμενες: κυρίως για τους ενσωματωμένους τύπους
  - Ρητές: χρήση του τελεστή μετατροπής (…)
  - Π.χ. `((float) M)/N`

- Δημιουργία αντικειμένων
  - Η δήλωση μίας μεταβλητής δεν δημιουργεί αντικείμενα
    - `Class point {double x; double y};`
  - Τα αντικείμενα δημιουργούνται μέσω του τελεστή new
    - `Point a = new Point(), b = new Point();`

# Δομικά στοιχεία


- Αναπαράσταση αντικειμένων
  - Στους ενσωματωμένους τύπους, το όνομα της μεταβλητής καθορίζει τη διεύθυνση όπου βρίσκεται η τιμή της μεταβλητής
  - Στα αντικείμενα, το όνομα είναι μία αναφορά (reference): καθορίζει τη διεύθυνση της θέσης μνήμης, που περιέχει τη διεύθυνση όπου βρίσκεται η πληροφορία
  - Οι αναφορές δείχνουν αντικείμενα στη μνήμη
    - `a.x = 1.0; a.y = 1.0; b.x = 4.0; b.y = 5.0;`
- Μεταβίβαση παραμέτρων
  - Οι ενσωματωμένοι τύποι μεταβιβάζονται με τιμή
    - Αντιγραφή της παραμέτρου σε τοπική μεταβλητή
  - Τα αντικείμενα μεταβιβάζονται με αναφορά
    - Αντιγραφή της αναφοράς σε τοπική μεταβλητή
    - Το αντικείμενο είναι προσπελάσιμο μέσω της αναφοράς

# Δομικά στοιχεία

- Παράδειγμα: η τάξη Point (για επεξεργασία γεωμετρικών δεδομένων)
  - Περιέχει δεδομένα και μεθόδους

```
class Point {
  double x, y;
  Point() { x = Math.random(); y = Math.random(); }
  Point(double x, double y) {this.x = x;this.y = y;}
  double r() { return Math.sqrt(x*x + y*y); }
  double theta() { return Math.atan2(y, x); }
  double distance(Point p) {
       double dx = x - p.x, dy = y - p.y;
       return Math.sqrt(dx*dx + dy*dy); }
  public String toString() {
       return "(" + x + "," + y + ")"; }
}
```

# Δομικά στοιχεία

- Υπερφόρτωση μεθόδων
  - Πολλές μέθοδοι με το ίδιο όνομα
  - Αλλά διαφορετικά signatures
  - Π.χ. 2 κατασκευάστριες μέθοδοι για την τάξη `Point`

- Χρήση των μεθόδων:
  - `Point c = new Point()` (δημιουργία ενός τυχαίου σημείου στον $R^2$)
  - `c.r()` (απόσταση του c από την αρχή των αξόνων)
  - `a.distance(b)` ή `b.distance(a)` (απόσταση μεταξύ του σημείου a και του σημείου b)

# Δομικά στοιχεία

- Κληρονομικότητα
  - Μας επιτρέπει να ορίζουμε νέους τύπους δεδομένων ξεκινώντας από ήδη υπάρχοντες
  - Μέλη βασικής κλάσης + μέλη εκτεταμένης κλάσης
  - Μπορούμε να επανορίσουμε μεθόδους της αρχικής κλάσης

```
class LabeledPoint extends Point {
  String id;
  void label(String name) { id = name; }
  public String toString() {
    return name + "("+x+","+y+")";
  }
}
```

# Πίνακες

- Πίνακας (array)
  - Η πιο θεμελιώδης δομή δεδομένων
  - Συλλογή δεδομένων ίδιου τύπου
  - Αποθηκεύονται σε μία σειρά από θέσεις
  - Άμεση πρόσβαση σε κάθε θέση μέσω αριθμοδείκτη
- Δήλωση και δημιουργία πίνακα
  - Οι δηλώσεις πινάκων δημιουργούν αναφορές (δεν δεσμεύεται μνήμη για αποθήκευση των στοιχείων του)
    - `int[] a; //ένας πίνακας τύπου int`
  - Η δημιουργία των πινάκων γίνεται μέσω του τελεστή new
    - `a = new int[10]; // δέσμευση μνήμης για πίνακα 10 θέσεων`
- Αναφορά στα στοιχεία του πίνακα
  - Χρήση του αριθμοδείκτη
    - `x=a[3]; // ανάγνωση της 4ης θέσης`
    - `a[2]=5; // εγγραφή της 3ης θέσης`

# Πίνακες

- Όρια πίνακα
  - Έστω ένας πίνακας a με N στοιχεία
  - Το πρώτο στοιχείο είναι το a[0]
  - Το τελευταίο στοιχείο είναι το a[n-1]
  - Αναφορές εκτός ορίων οδηγούν σε εξαίρεση
- Επεξεργασία όλων των στοιχείων
  - Δεν χρειάζεται να γνωρίζω το μήκος
  - Τα αντικείμενα πινάκων έχουν το πεδίο length

```
for (int i=0; i < a.length; i++)
  {  …a[i]… }
```

  - Προσοχή: το τελευταίο στοιχείο είναι στο a.length – 1
  - Χρήση τοπικής μεταβλητής που δηλώνεται επί τόπου

# Πίνακες

- Δήλωση και δημιουργία πινάκων

```
int c[] = new int [12];
int c[];
c = new int [12];
```

- Δημιουργία διαφόρων πινάκων σε μία δήλωση

```
String b[] = new String[100], x[] = new String[27];
String b[] = new String[100];
String x[] = new String[27];
```

- Οι αγκύλες μπορεί να βρίσκονται σε δύο σημεία

```
double[] array1; double[] array2;
double array1[]; double array2[];
double[] array1, array2;
```

# Πίνακες

Πότε υπερτερεί η χρήση πινάκων σε σχέση με άλλες δομές δεδομένων (π.χ. λίστες) ?

1. Όταν γνωρίζουμε εκ των προτέρων τον αριθμό των αντικειμένων που έχουμε να επεξεργαστούμε
2. Όταν θέλουμε η προσπέλαση κάθε στοιχείου να γίνεται πολύ γρήγορα

# Το κόσκινο του Ερατοσθένη (3ος αιώνας π.Χ.)

- Πρώτοι αριθμοί
  - Ακέραιοι $X \geq 2$ που διαιρούνται μόνο με τον X και το 1, π.χ. 2,3,5,7,11
  - Εύρεση μέσω του αλγορίθμου του Ερατοσθένη
- Το κόσκινο (sieve) του Ερατοσθένη
  - Είσοδος: Ένας ακέραιος $N \geq 2$
  - Στόχος: εύρεση όλων των πρώτων $< N$
  - Δημιουργία πίνακα N στοιχείων boolean
    - Αρχικά όλα τα στοιχεία είναι true
    - Σταδιακά οι μη πρώτοι θα γίνουν false
  - Ξεκινάμε με τη θέση 2
  - Για κάθε στοιχείο του πίνακα
    - Αν είναι false (μη πρώτος), προχωράμε στο επόμενο
    - Αν είναι true βρες όλα τα πολλαπλάσιά του
    - Θέσε κάθε πολλαπλάσιο (που είναι $< N$) σε false

# Το κόσκινο του Ερατοσθένη

```
class Primes {
public static void main(String[] args) {
     int N = Integer.parseInt(args[0]);
     boolean[] a = new boolean[N];
     for (int i = 2; i < N; i++) a[i] = true;
     for (int i = 2; i < N; i++)
          if (a[i])
             for (int j = i; j*i < N; j++)
                a[i*j] = false;// πολλαπλάσια του i
                               // δεν είναι πρώτοι
     for (int i = 2; i < N; i++)
        if (a[i]) Out.print(" " + i);
     Out.println();
     }
}
```

# Το κόσκινο του Ερατοσθένη

- Παράδειγμα εκτέλεσης
  - Οι πρώτοι αριθμοί μέχρι το 32
  - Αρχικά όλες οι τιμές είναι true
- Ερωτήσεις (επαγωγικές αποδείξεις)
  1. Γιατί αγνοούμε τα στοιχεία που είναι false;
     - Κάθε αριθμός γράφεται ως γινόμενο πρώτων αριθμών (θεώρημα Ευκλείδη). Αρκεί να φιλτράρουμε τα πολ/σια των πρώτων.
  2. Γιατί ο βρόχος ξεκινάει από το $i^2$;
     - Αν `x<i*i`, και το `x` δεν είναι πρώτος, τότε το `x` έχει κάποιον πρώτο παράγοντα `<i-1`. Άρα το `a[x]` έγινε false σε κάποια προηγούμενη επανάληψη

| i | 2 | 3 | 5 | a[i] |
|---|---|---|---|---|
| 2 | 1 | | | 1 |
| 3 | 1 | | | 1 |
| 4 | 1 | 0 | | |
| 5 | 1 | | | 1 |
| 6 | 1 | 0 | | |
| 7 | 1 | | | 1 |
| 8 | 1 | 0 | | |
| 9 | 1 | | 0 | |
| 10 | 1 | 0 | | |
| 11 | 1 | | | 1 |
| 12 | 1 | 0 | 0 | |
| 13 | 1 | | | 1 |
| 14 | 1 | 0 | | |
| 15 | 1 | | 0 | |
| 16 | 1 | 0 | | |
| 17 | 1 | | | 1 |
| 18 | 1 | 0 | 0 | |
| 19 | 1 | | | 1 |
| 20 | 1 | 0 | | |
| 21 | 1 | | 0 | |
| 22 | 1 | 0 | | |
| 23 | 1 | | | 1 |
| 24 | 1 | 0 | 0 | |
| 25 | 1 | | 0 | |
| 26 | 1 | 0 | | |
| 27 | 1 | | 0 | |
| 28 | 1 | 0 | | |
| 29 | 1 | | | 1 |
| 30 | 1 | 0 | 0 | 0 |
| 31 | 1 | | | 1 |

# Εξαιρέσεις

- Εύρωστη κατανομή πίνακα
  - Σε περίπτωση ανεπαρκούς μνήμης έχουμε εξαίρεση
  - Ο κώδικας πρέπει να τροποποιηθεί ως εξής

```
boolean[] a;
try { a = new boolean[N]; }
catch (OutOfMemoryError e) {
  Out.println("Out of memory"); return; }
```

- Στις εργασίες σας να φροντίζετε πάντα να έχετε ελέγχους για εξαιρέσεις

# Χρόνος Εκτέλεσης

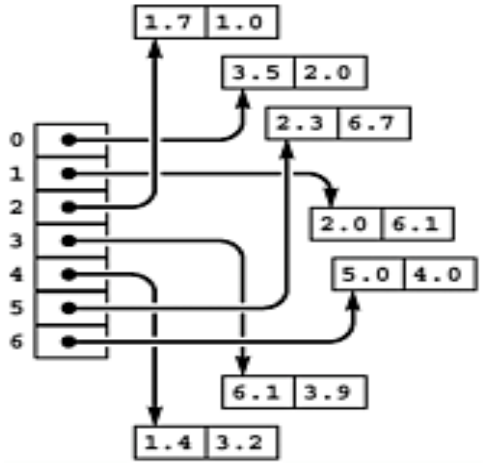
- Πόσο χρόνο χρειάζεται το πρόγραμμα?
- Μία ακριβής απάντηση χρειάζεται αποτελέσματα από τη Θεωρία Αριθμών για την πυκνότητα των πρώτων.
- Προσεγγιστικά:

$$N + N/2 + N/3 + N/5 + N/7 + \ldots \leq N + N/2 + N/3 + N/4 + N/5 + \ldots$$
$$= N(1+1/2+1/3+1/4+\ldots)$$

- Αποδεικνύεται ότι $1+1/2+1/3+\ldots \approx \ln N$
- Άρα συνολικά περίπου $N \ln N$ (ίδια τάξη μεγέθους με $N \log N$)

# Πίνακες Αντικειμένων

- Μπορούμε να έχουμε πίνακες από αντικείμενα, αντί για ακεραίους ή άλλους στοιχειώδεις τύπους
  - `Point[] a = new Point[N]`
- Σε ένα πίνακα ακεραίων, το `a[i]` περιέχει την τιμή του ακεραίου
- Σε πίνακα αντικειμένων, το `a[i]` περιέχει την αναφορά στη διεύθυνση μνήμης που βρίσκεται το αντικείμενο




```
a[3].x = 6.1
```

# Εύρεση Πλησιέστερων Σημείων

- Παράδειγμα χρήσης πίνακα αντικειμένων Point
- Είσοδος: Συνολικό πλήθος σημείων N, απόσταση d
- Έξοδος: Πόσα ζεύγη από τα σημεία απέχουν < d

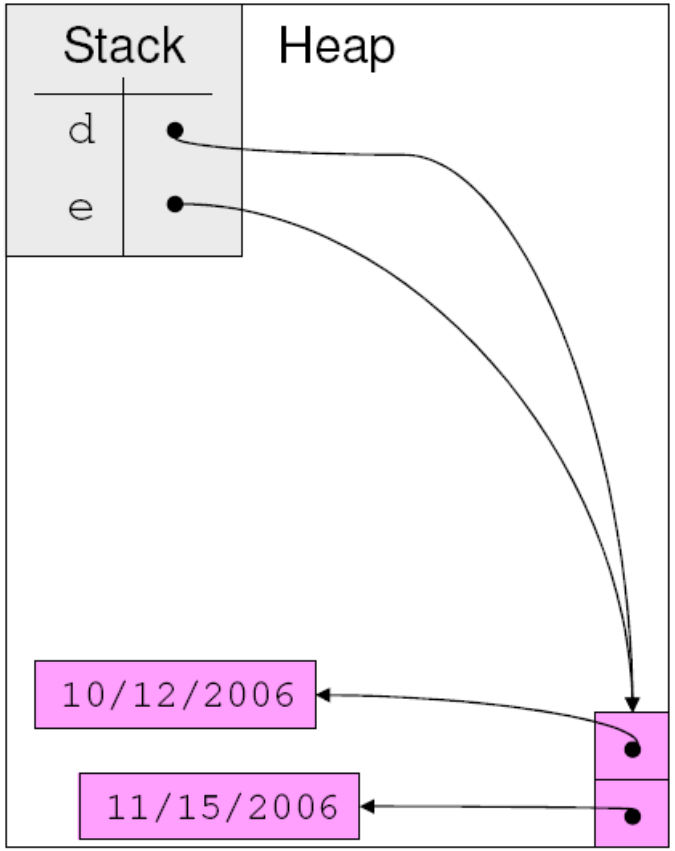
```
class ClosePoints {
  public static void main(String[] args) {
        int cnt = 0, N = Integer.parseInt(args[0]);
        double d = Double.parseDouble(args[1]);
        Point[] a = new Point[N];
        for (int i = 0; i < N; i++) a[i]=new Point();
        for (int i = 0; i < N; i++)
           for (int j = i+1; j < N; j++)
              if (a[i].distance(a[j])<d) cnt++;
        Out.print(cnt + " pairs ");
  Out.println("closer than " + d);
} }
```

# Αντιγραφή πινάκων

- Ρηχή αντιγραφή: αντιγραφή αναφοράς

```
Date[] d = {
   new
   Date(12,10,2006),
   new
   Date(15,11,2006)
};

Date[] e = d;
```

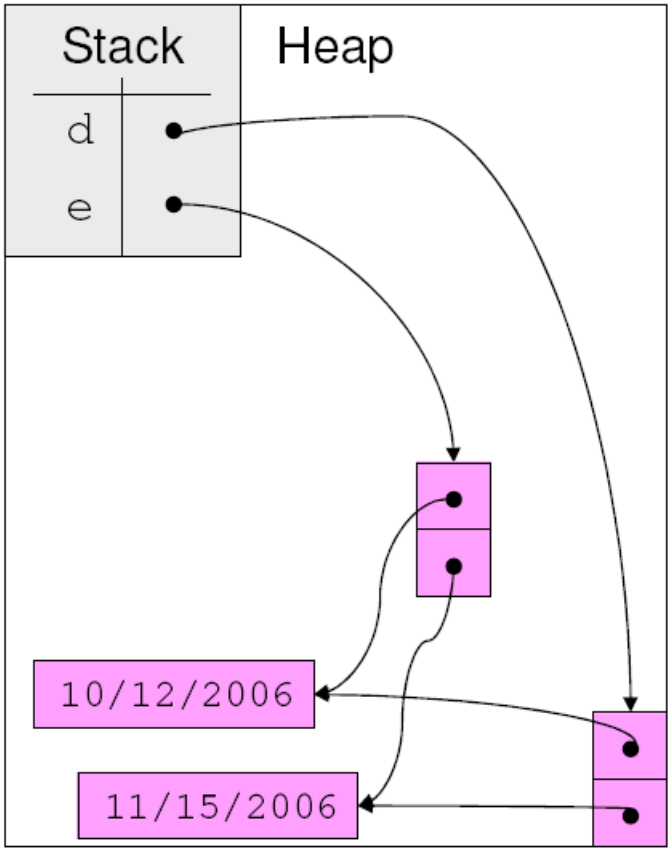


# Αντιγραφή πινάκων

- Ημι-βαθιά αντιγραφή: αντιγραφή αναφορών πίνακα

```
Date[] d = {
   new Date(12,10,2006),
   new Date(15,11,2006)
};

Date[] e = new
   Date[d.length];
for (int i=0; i <
   d.length, i++)
   e[i] = d[i];
```

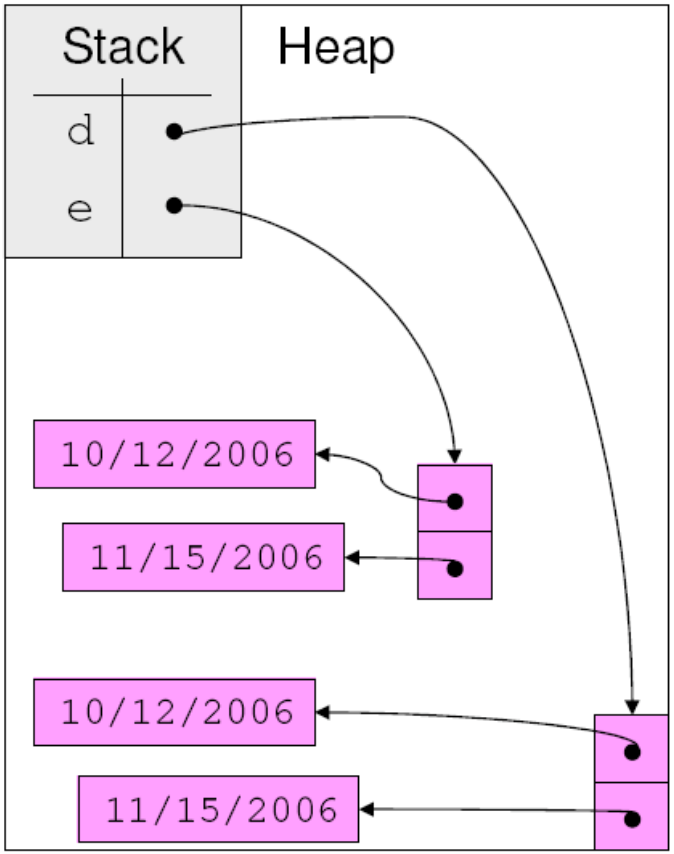


# Αντιγραφή πινάκων

- Βαθιά αντιγραφή: αντιγραφή αντικειμένων πίνακα

```
Date[] d = {
   new Date(12,10,2006),
   new Date(15,11,2006)
};

Date [] e = new
   Date[d.length];
for (int i = 0; i <
   d.length; i++)
   e[i] = new Date(d);
```



# Αλλαγή μεγέθους πίνακα

- Πώς αλλάζουμε το μέγεθος ενός πίνακα;
  - Το μέγεθος κάθε πίνακα είναι αμετάβλητο
  - Μόνη λύση η αντιγραφή σε άλλο πίνακα
    - Προσοχή στο κόστος της αντιγραφής
- Αντιγραφή σε πίνακα διαφορετικού μεγέθους
  - Δημιουργία πίνακα με το επιθυμητό μέγεθος
  - Αντιγραφή περιεχομένων σε νέο πίνακα
  - Αλλαγή αναφοράς στον νέο πίνακα

```
int[] a = new int[5];
...
int[] temp = new int[a.length+1];
for (int i=0; i < a.length; i++)
  temp[i]=a[i];
a = temp;
```

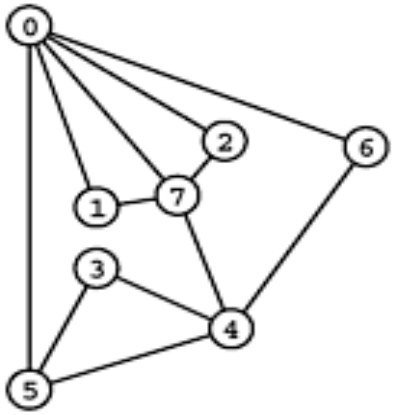
# Πολυδιάστατοι Πίνακες

- Πολλές εφαρμογές απαιτούν τη χρήση περισσότερων διαστάσεων π.χ. απεικόνιση σημείων στον τρισδιάστατο χώρο
- Διδιάστατοι πίνακες:
  - `double[][] c = new double[N][M];`
  - Αποθήκευση στη μνήμη γίνεται σειριακά, πρώτα η $1^{η}$ γραμμή, μετά η $2^{η}$ κ.ο.κ.
  - Επεξεργασία των στοιχείων συνήθως με διπλό βρόχο

```
for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) {
        ...
        //επεξεργασία του c[i][j]
    }
}
```

# Παράδειγμα: 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 |

# Ειδικές Κατηγορίες Πινάκων

- Αραιοί (sparse) πίνακες: Πίνακες όπου μεγάλο ποσοστό των στοιχείων είναι 0 (π.χ. αραιοί γράφοι)
- Έστω ο πίνακας:

| | | | |
|---|---|---|---|
| 0 | 7 | 0 | 0 |
| 1 | 2 | 0 | 0 |
| 0 | 0 | 4 | 0 |
| 9 | 0 | 0 | 0 |

- Σπατάλη μνήμης η αποθήκευση ως 2διάστατο πίνακα
- Μπορούμε να χρησιμοποιήσουμε μονοδιάστατο πίνακα
  - Κάθε μη μηδενικό στοιχείο αποθηκεύεται ως μία τριάδα (συντεταγμένες στον πίνακα + τιμή)
  - 1 2 7 2 1 1 2 2 2 3 3 4 4 1 9

# Ειδικές Κατηγορίες Πινάκων

- Τριγωνικοί πίνακες: Πίνακες όπου πάνω ή κάτω από τη διαγώνιο όλα τα στοιχεία είναι 0
- π.χ.:

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 9 & 0 & 0 & 0 \end{matrix}$$

(κάτω τριγωνικός)

- Υπάρχουν πιο αποδοτικοί μέθοδοι αποθήκευσης με μονοδιάστατο πίνακα
- Η χρήση εναλλακτικών μεθόδων αποθήκευσης εξαρτάται από:
  - Πόσο σημαντική είναι η εξοικονόμηση μνήμης για την εφαρμογή
  - Τι άλλες λειτουργίες χρειάζεται να εκτελούμε στον πίνακα

# Άλλες Υλοποιήσεις

- Αρκετές μέθοδοι για πίνακες είναι διαθέσιμες στην βιβλιοθήκη της Java, μέσω της κλάσης Vector
- Πιο ευέλικτη δομή από τους πίνακες (ένα αντικείμενο Vector μπορεί να αλλάζει μεγεθος)
- Προσπέλαση των στοιχείων είναι πιο αργή (μέσω της μεθόδου get() )
- Για καλύτερη αποδοτικότητα και απλότητα του κώδικα, θα αποφεύγουμε να χρησιμοποιούμε έτοιμες μεθόδους