CS1110 21 April 2011 Ragged arrays

Reading for today: sec. 9.3.
Reading for next time: chapter 16, applications and applets

1. Slow to reveal!

```
/** Extract and return … */
public String reveal() {
  …

  int p= 4;
  String result= "";

  // inv: All hidden chars before
  // pixel p are in result[0..k-1]
  for (int k= 0; k < len; k= k+1) {
    result= result +
          (char) (getHidden(p));
    p= p+1;
  }
  return result;
}
```

gives n^2 algorithm (n is message length)

```
/** Extract and return … */
public String reveal() {
  …

  int p= 4;
  char[] result= new char[len];

  // inv: All hidden chars before
  // pixel p are in result[0..k-1]
  for (int k= 0; k < len; k= k+1) {
    result[k]=
          (char) (getHidden(p));
    p= p+1;
  }
  return new String(result);
}
```

linear algorithm

Review of two-dimensional arrays

Type of d is **int**[][]

("**int** array array"/ "an array of **int** arrays")

To declare variable d:

int d[][];

To create a new array and assign it to d:

d= **new int**[5][4];

or, using an array initializer,

d= **new int**[][]{ {5,4,7,3}, {4,8,9,7}, {5,1,2,3}, {4,1,2,9}, {6,7,8,0} };

d

	0	1	2	3
0	5	4	7	3
1	4	8	9	7
2	5	1	2	3
3	4	1	2	9
4	6	7	8	0

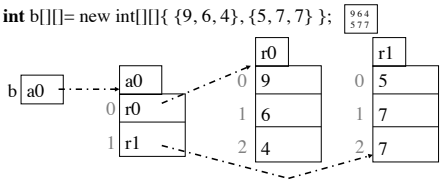
Some mysteries: an odd asymmetry, and strange toString output (see demo).

Number of rows of d: d.length

Number of columns in row r of d: d[r].length

How multi-dimensional arrays are stored: arrays of arrays

int b[][]= new int[][]{ {9, 6, 4}, {5, 7, 7} };



b holds the name of a one-dimensional array object with b.length elements; its elements are the names of 1D arrays.

b[i] holds the name of a 1D array of **int**s of length b[i].length

java.util.Arrays.deepToString recursively creates an appropriate String.

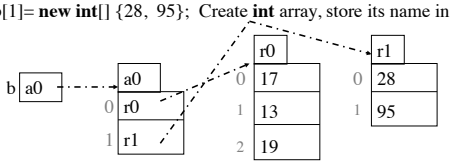
Ragged arrays: rows have different lengths

int[][] b; Declare variable b of type **int**[][]

b= **new int**[2][] Create a 1-D array of length 2 and store its name in b. Its elements have type **int**[] (and start as **null**).

b[0]= **new int**[] {17, 13, 19}; Create **int** array, store its name in b[0].

b[1]= **new int**[] {28, 95}; Create **int** array, store its name in b[1].



Application: recommender systems

Large collections of *association data* abound, but often, many possible associations have the default value, so the data is *sparse.*

Netflix data: (user, movie, score): 480K × 18K = 8.6B possible scores to track, but there are only (!) 100M actual scores.

GroupLens data (freely distributed by U. Minn): the small set has 943×1682= 1.5M possibilities, but only 100K actual scores.

How might Netflix, Amazon, etc. use this kind of association data to generate recommendations?

1. Represent each user by an array of movie ratings
2. Find similar users according to the similarity of the corresponding arrays, and report their favorite movies

This seems to suggest a 2-D, user-by-movie array.

Recommender-system application (cont.)

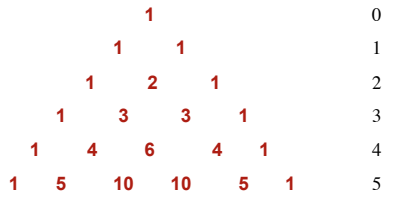
GroupLens data (freely distributed by U. Minn): the small set has 943×1682= 1.5M possibilities, but only 100K actual scores.

Main idea:
For each user, DON'T store an **int** array of length 1682;
store a movie-sorted array of objects corresponding to the ratings for just the movies that user saw (avg. length: 59!).

This means a 2-D *ragged* user/movie array.

Another very useful technique (among many more substantive ones; take more CS courses!): map the movie/rater names to ints, b/c they can be meaningful array indices.

Pascal's Triangle

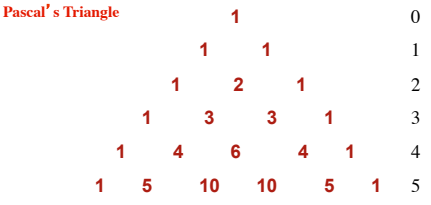


The first and last entries on each row are 1.

Each other entry is the sum of the two entries above it

row r has r+1 values.

Pascal's Triangle

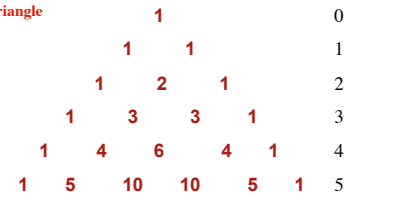


Entry p[i][j] is the number of ways i elements can be chosen from a set of size j !

p[i][j] = "i choose j" = $\binom{i}{j}$

recursive formula:
for $0 < i < j$, $p[i][j] = p[i-1][j-1] + p[i-1][j]$

Pascal's Triangle



Binomial theorem: Row r gives the coefficients of $(x + y)^r$

$(x + y)^2 = 1x^2 + 2xy + 1y^2$

$(x + y)^3 = 1x^3 + 3x^2y + 3xy^2 + 1y^3$

$(x + y)^r = \sum_{0 \le k \le r}$ (k choose r) $x^k y^{r-k}$

Function to compute first r rows of Pascal's Triangle in a ragged array

```
/** =ra gged array of first n rows of Pascal' s triangle.
    Precondition: 0 ≤ n */
public static int[][] pascalTriangle(int n) {
    int[][] b= new int[n][];     // First n rows of Pascal's triangle
    // invariant: rows 0..i-1 have been created
    for (int i= 0; i != b.length; i= i+1) {
      // Create row i of Pascal's triangle
        b[i]= new int[i+1];

      // Calculate row i of Pascal's triangle
        b[i][0]= 1;
        // invariant b[i][0..j-1] have been created
        for (int j= 1; j < i; j= j+1) {
          b[i][j]= b[i-1][j-1] + b[i-1][j];
        }
        b[i][i]= 1;
    }
    return b;
  }
```