Monday March 12

Lecture 9

Draw the flow chart for a nested loop:

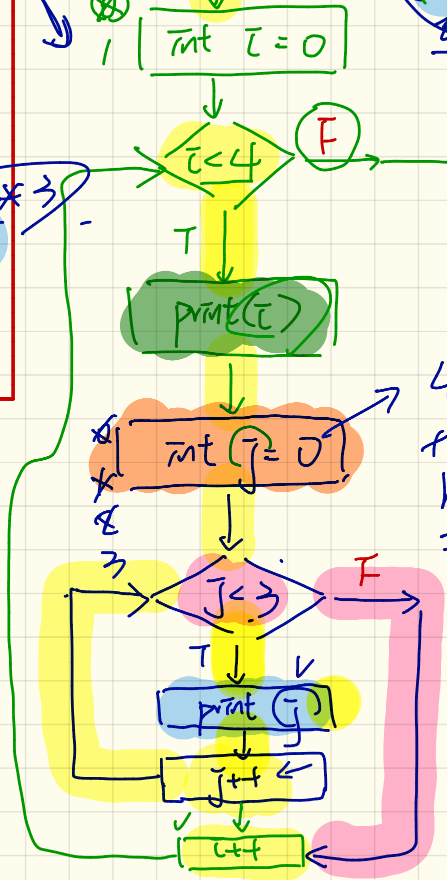
outer loop (4 iterations 0, 1, 2, 3)

inner loop (3 iterations 0, 1, 2)

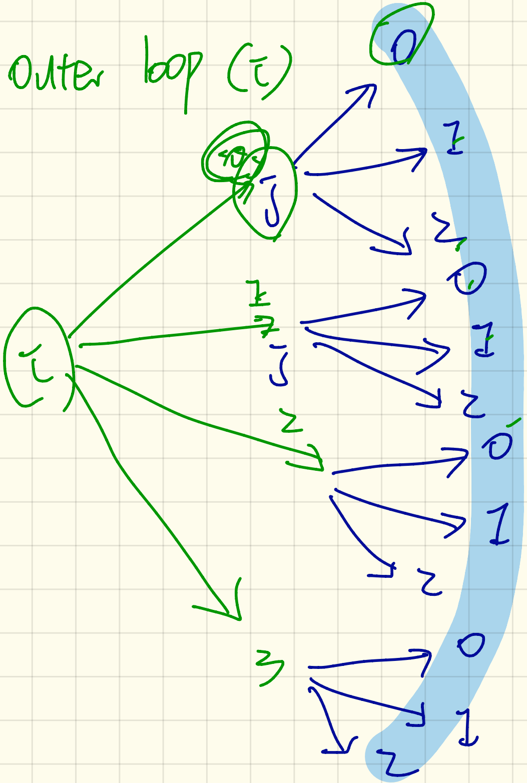
```
outer
→1 for(int i = 0; i<4 ; i++) {
2       print(i);            → executed 4 times
3       for(int j=0; j<3 ; j++){      4*3
4           print(j);        → executed 12 times.
        }
   }
```

outputs

```
0          2
 0.         0.
 1.         1.
 2.         2.
1          3
 0.         0.
 1.         1.
 2.         2.
```



4 times that we re-initialize value of j => does not persist between iterations of outer loop.



has duplicate ?

a2

0	1	2	3
1	2	3	4

False

✓

a

0	1	2	3
1	4	2	4

true

a[1] != a[3] → witness

a →

0	1	2	3
1	4	2	4

4 elements
↳ 4 * 4

16 comparisons
↓
4 cases not to be considered.

a[0] == a[0] ✓ → we should not consider this as a witness of duplicate

a[0] == a[1] ✗

a[0] == a[2] ✗

a[0] == a[3] ✗

a[1] == a[0] ✗

a[1] == a[1] ✓ →

a[1] == a[2] ✗

a[1] == a[3] ✓ witness!

duplicates:

$a[i] == a[j]$ &&

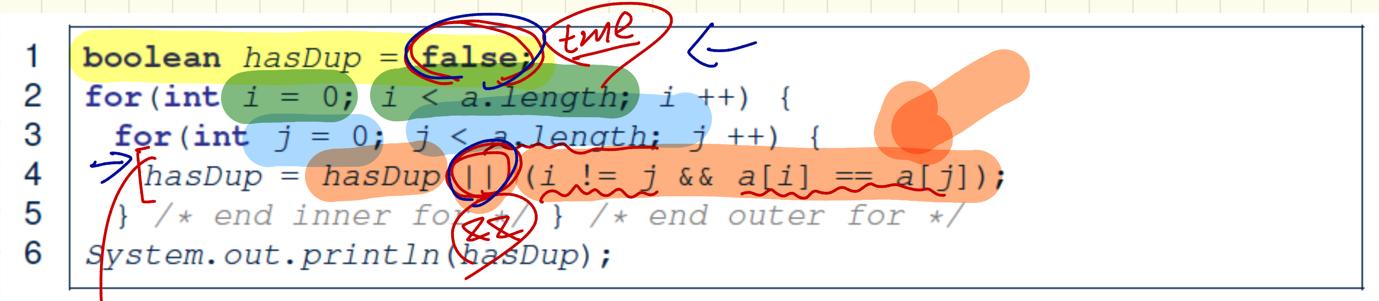
$i != j$

don't consider as a witness

a[2]

a[3]

Correct but Redundant Scan

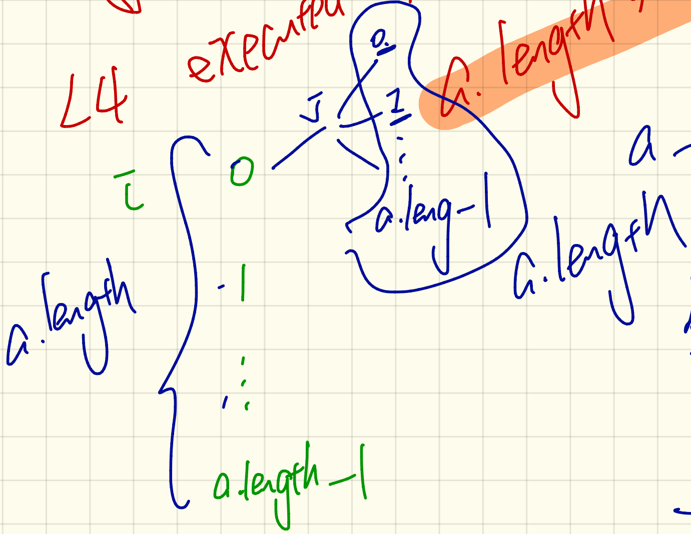


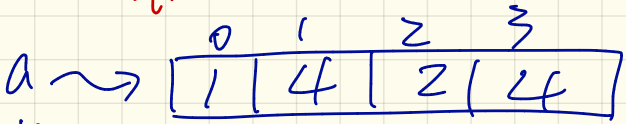
```
1 boolean hasDup = false;
2 for(int i = 0; i < a.length; i ++) {
3   for(int j = 0; j < a.length; j ++) {
4     hasDup = hasDup || (i != j && a[i] == a[j]);
5   } /* end inner for */ } /* end outer for */
6 System.out.println(hasDup);
```

true

&&

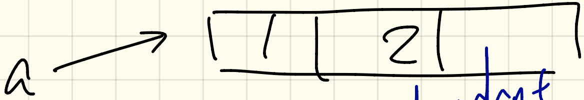
L4 executed for a.length * a.length times



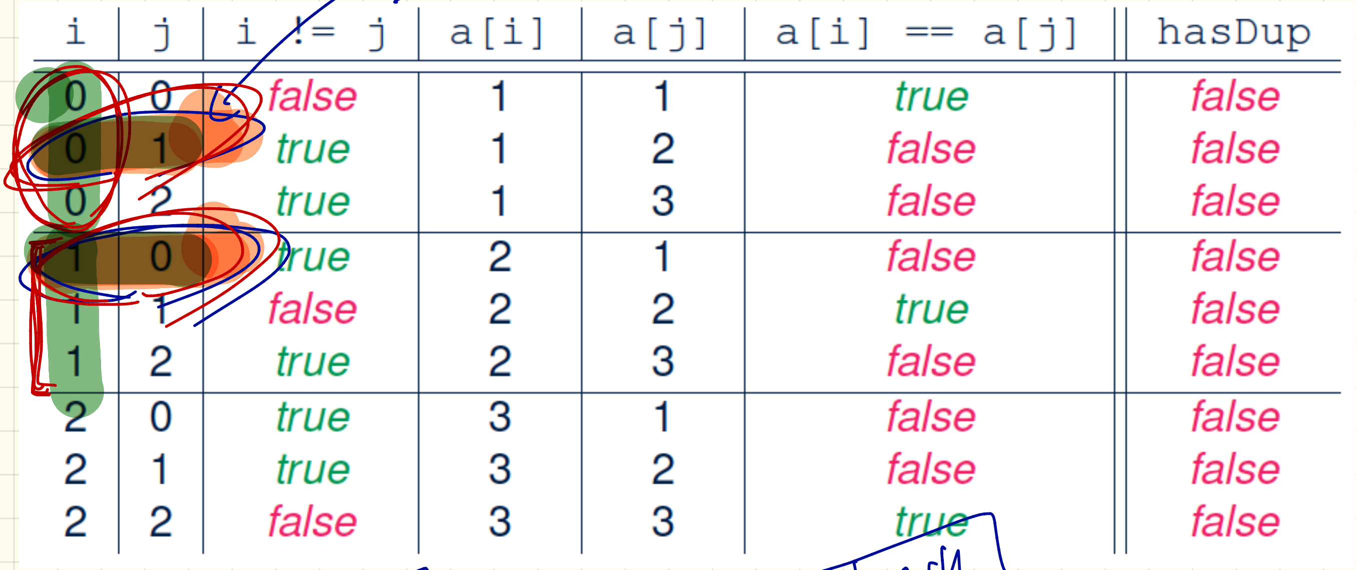


before the change : True (∵ a[1] == a[3])

after the change : False (∵ a[0] == a[1] is false)



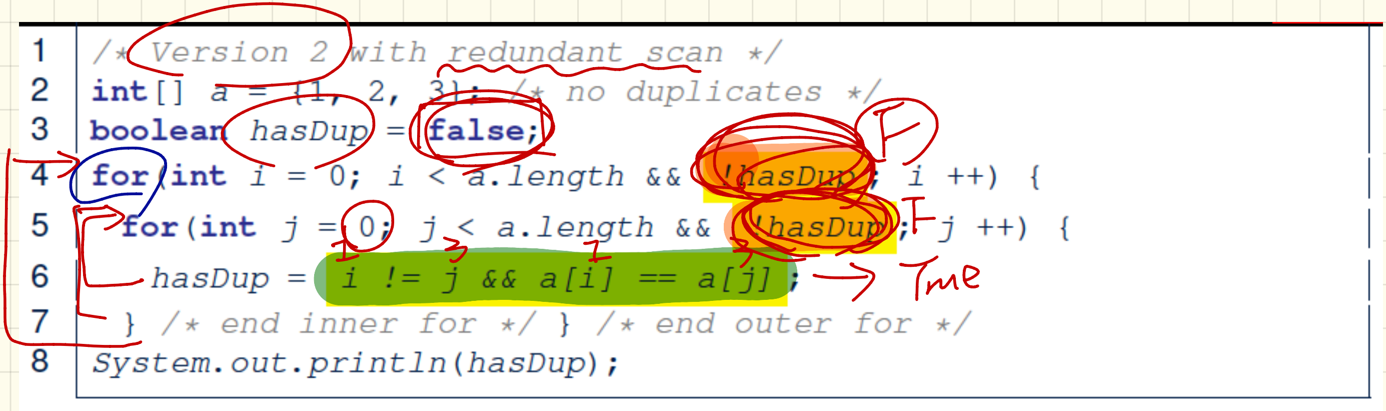
redundant.



i	j	i != j	a[i]	a[j]	a[i] == a[j]	hasDup
0	0	false	1	1	true	false
0	1	true	1	2	false	false
0	2	true	1	3	false	false
1	0	true	2	1	false	false
1	1	false	2	2	true	false
1	2	true	2	3	false	false
2	0	true	3	1	false	false
2	1	true	3	2	false	false
2	2	false	3	3	true	false

Efficiency: $\underbrace{a.length}_{n} * \underbrace{a.length}_{n}$

n^2.

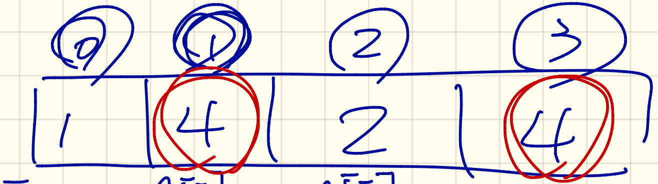


```
/* Version 2 with redundant scan */
int[] a = {1, 2, 3}; /* no duplicates */
boolean hasDup = false;
for(int i = 0; i < a.length && !hasDup; i ++) {
  for(int j = 0; j < a.length && !hasDup; j ++) {
    hasDup = i != j && a[i] == a[j] ;
} /* end inner for */ } /* end outer for */
System.out.println(hasDup);
```

F

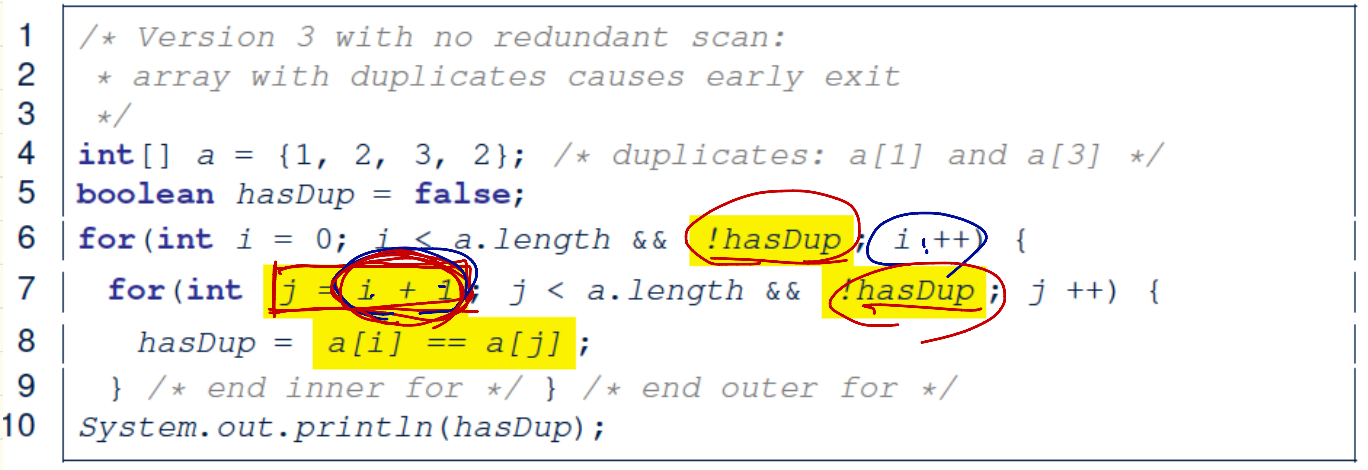
F

True



i	j	a[i] == a[j]		hasDup
0	0	T	(ignore)	~~T~~ F
0	1	F		
0	2	F		
0	3	F		
1	0	F		
1	1	T	(ignore)	
1	2	F		
1	3	T		T

redundant scan still there



```
/* Version 3 with no redundant scan:
 * array with duplicates causes early exit
 */
int[] a = {1, 2, 3, 2}; /* duplicates: a[1] and a[3] */
boolean hasDup = false;
for(int i = 0; i < a.length && !hasDup; i ++) {
  for(int j = i + 1; j < a.length && !hasDup; j ++) {
    hasDup = a[i] == a[j];
  } /* end inner for */ } /* end outer for */
System.out.println(hasDup);
```

I	J
0	1
0	2
0	3
1	2
1	3
2	3
~~3~~	

0	1	2	3
1	4	2	4

ok to miss these

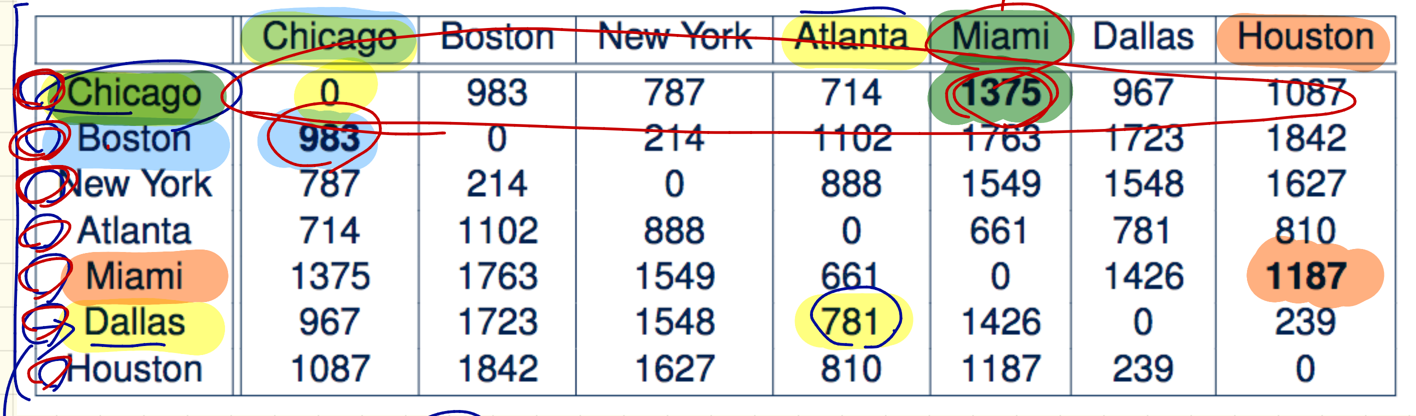
missing from versions 1 & 2 :

1	0	(covered)
1	1	(don't care)

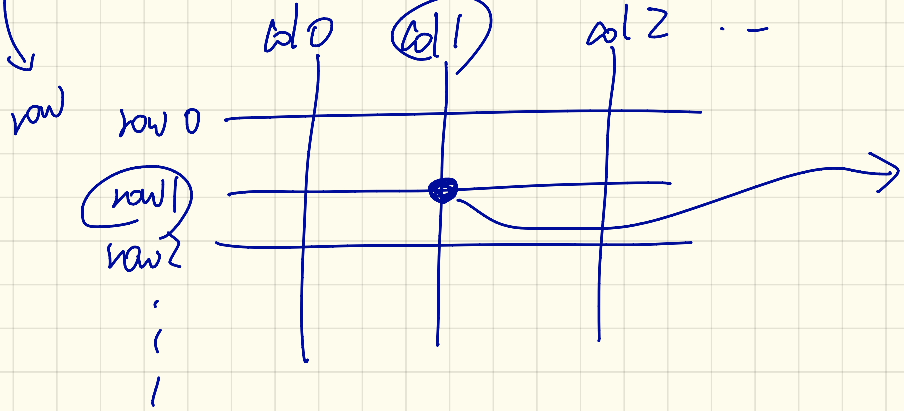
Column → { Bos, Chi, MIA, Hou }

983 1375 1187

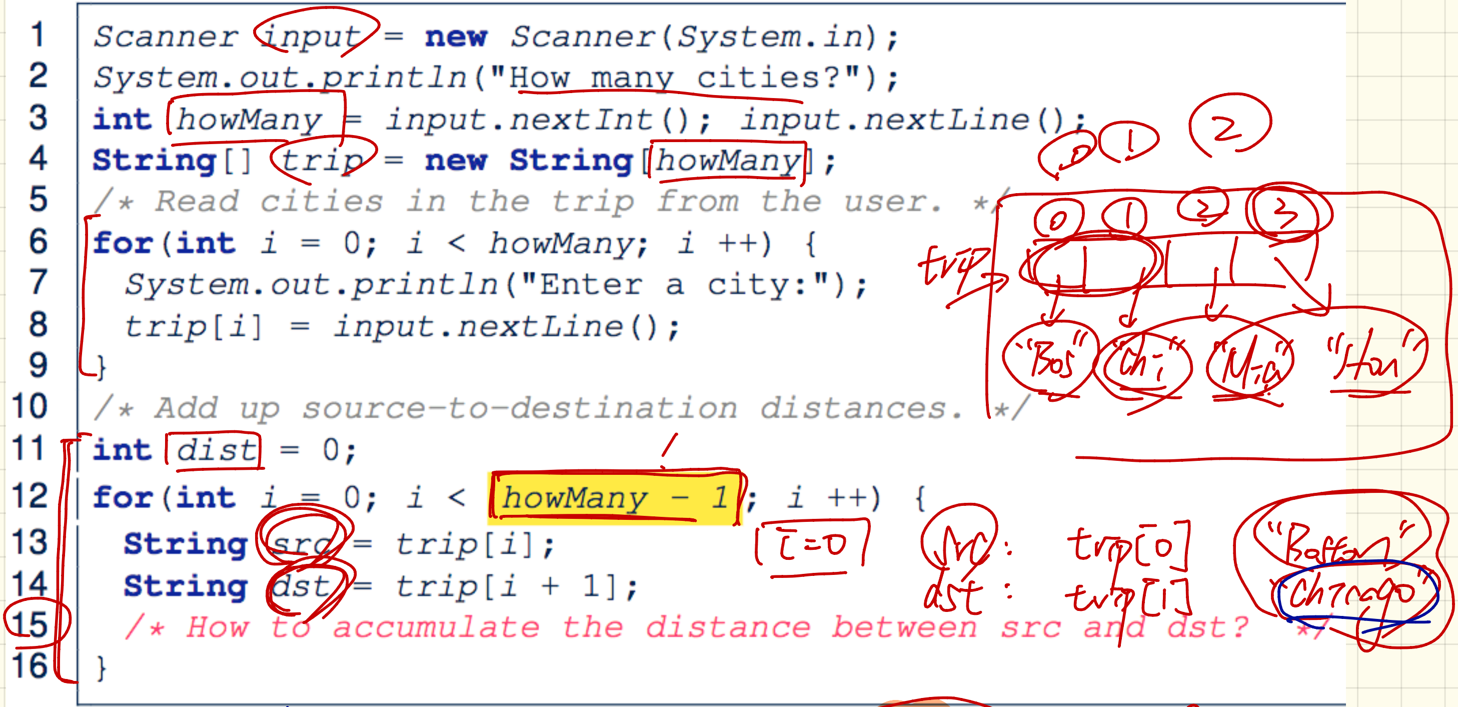
0



	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0



intersection of row 1 and col 1



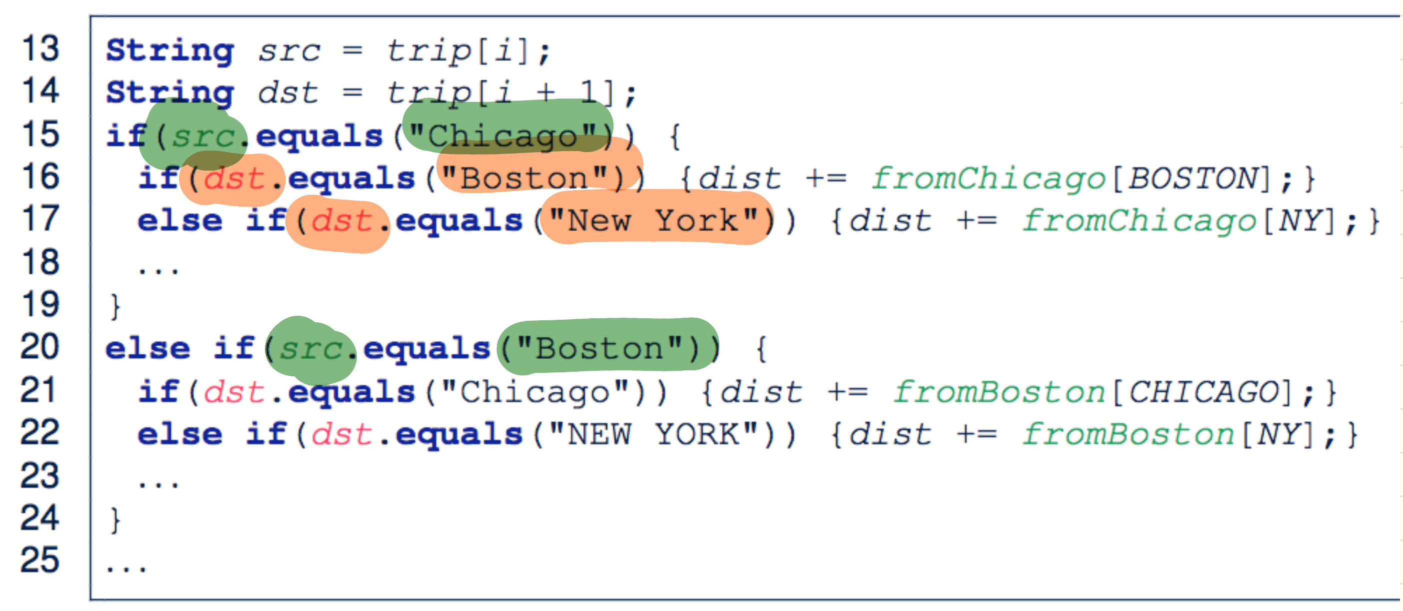
```
Scanner input = new Scanner(System.in);
System.out.println("How many cities?");
int howMany = input.nextInt(); input.nextLine();
String[] trip = new String[howMany];
/* Read cities in the trip from the user. */
for(int i = 0; i < howMany; i ++) {
  System.out.println("Enter a city:");
  trip[i] = input.nextLine();
}
/* Add up source-to-destination distances. */
int dist = 0;
for(int i = 0; i < howMany - 1; i ++) {
  String src = trip[i];
  String dst = trip[i + 1];
  /* How to accumulate the distance between src and dst? */
}
```

if (src. equals("Boston")) {
  if (dst. equals ("chicago")) {
    fromBoston [CHICAGO] ;
  }
}

fromBoston ["chicago"]

depending on value of src, determine which array.

depending on val of dst, figure out index



```
String src = trip[i];
String dst = trip[i + 1];
if(src.equals("Chicago")) {
  if(dst.equals("Boston")) {dist += fromChicago[BOSTON];}
  else if(dst.equals("New York")) {dist += fromChicago[NY];}
  ...
}
else if(src.equals("Boston")) {
  if(dst.equals("Chicago")) {dist += fromBoston[CHICAGO];}
  else if(dst.equals("NEW YORK")) {dist += fromBoston[NY];}
  ...
}
...
```

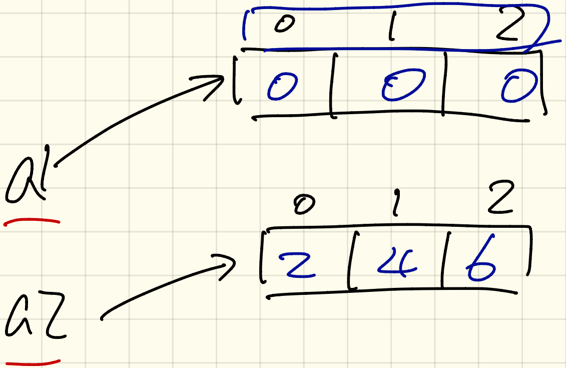
1D - array

index

int

size

```
int[] a1 = new int[3];
int[] a2 = {2, 4, 6};
```



2D- array

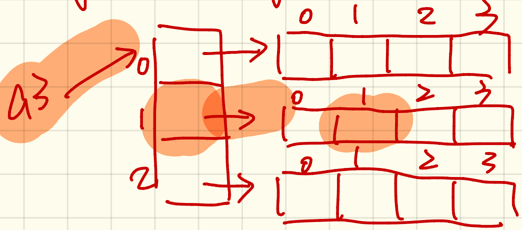
points to an array of 3 elements, each of which being an array of 4 elements.

# of rows

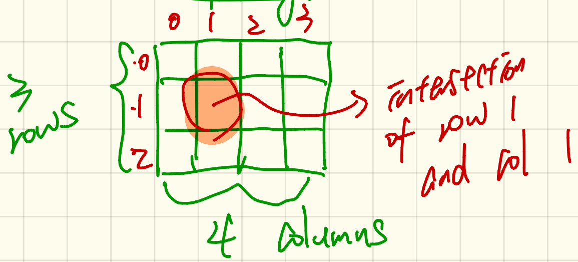
# of columns.

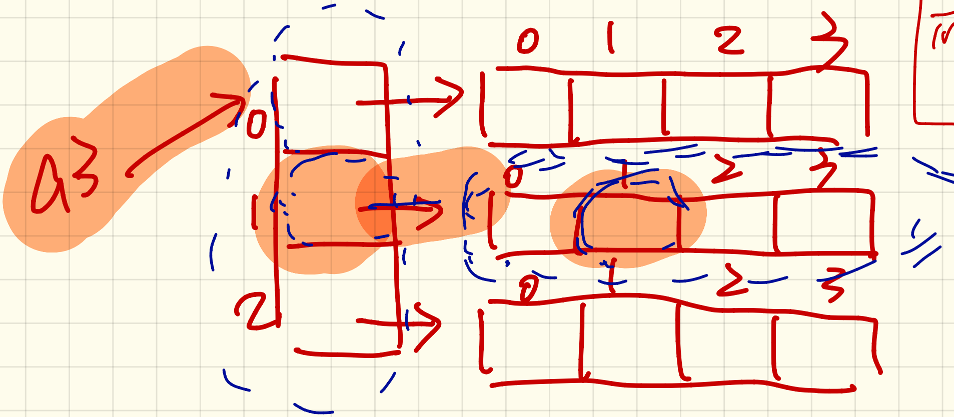
```
int[][] a3 = new int[3][4];
```

programmatically



Conceptually





```
int[][][][]...[]
        a =
  new int[2][3][4]..[10]
```

a3 : an array of arrays of integers.

a3[1] : an array of integers

a3[1][1] : an integer.

Conceptually

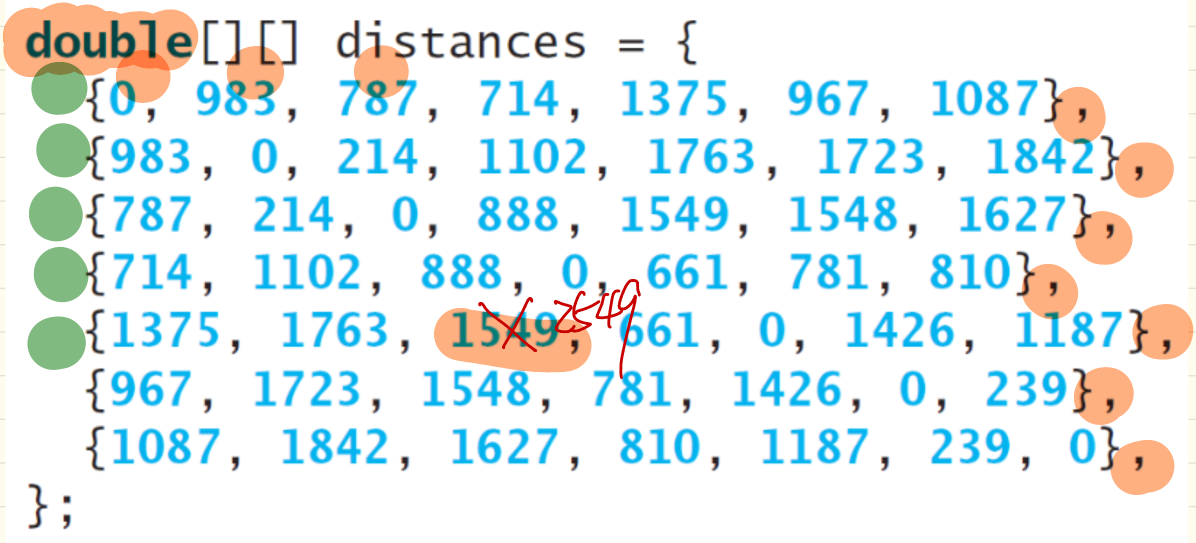
	0	1	2	3
0	1	8	5	11
1	6	3	9	2
2	4	10	7	12

Approach 1

```
int[][] a1 = new int[3][4];
a1[0][0] = 1;
   ⋮
```

Approach 2

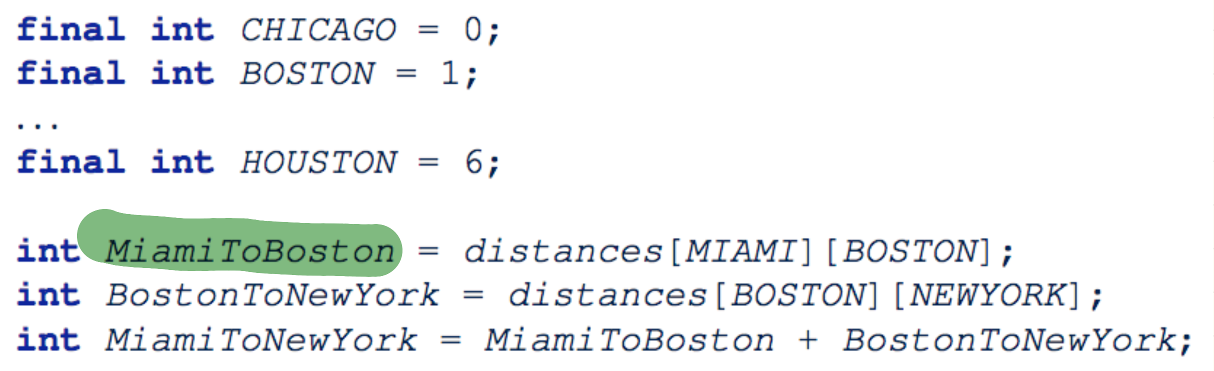
```
int[][] a2 = {
   {1, 8, 5, 11},
   {6, 3, 9, 2},
   {4, 10, 7, 12}
};
```



```
double[][] distances = {
  {0, 983, 787, 714, 1375, 967, 1087},
  {983, 0, 214, 1102, 1763, 1723, 1842},
  {787, 214, 0, 888, 1549, 1548, 1627},
  {714, 1102, 888, 0, 661, 781, 810},
  {1375, 1763, ~~1549~~ 2549, 661, 0, 1426, 1187},
  {967, 1723, 1548, 781, 1426, 0, 239},
  {1087, 1842, 1627, 810, 1187, 239, 0}
};
```

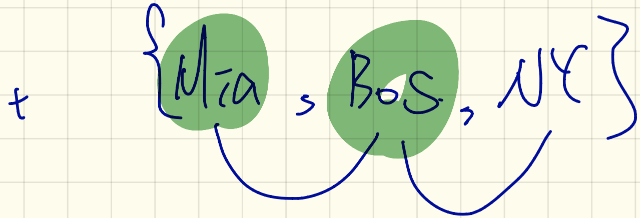
Q: println(distances[4][2])? → 1549

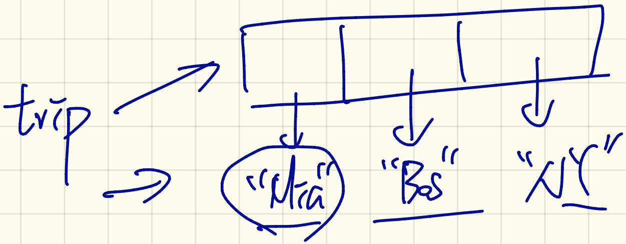
distances[4][2] = 2549;



```
final int CHICAGO = 0;
final int BOSTON = 1;
...
final int HOUSTON = 6;

int MiamiToBoston = distances[MIAMI][BOSTON];
int BostonToNewYork = distances[BOSTON][NEWYORK];
int MiamiToNewYork = MiamiToBoston + BostonToNewYork;
```

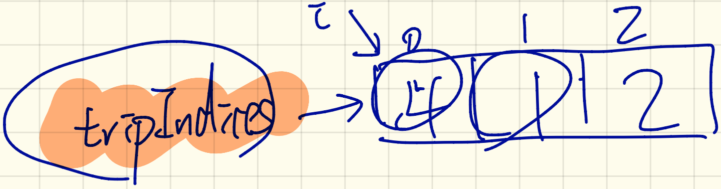




int src=tripIndices[0];

int dst=tripIndices[1];

distances[src][dst]


i
0
1
2
tripIndices
4
1
2