

مصفوفات 1D + 2D
قوانين، شرح وأسئلة مع الحل

1D قوانين مصفوفة أحادية

int[]

`int[] a = new int[5];` 5 خانات، وكلهن 0	`int[] a = {4, 1, 7};` بناء مع القيم، والطول 3
`a.length` عدد الخانات. بدون أقواس، مش length()	`a[0]` أوّل خانة: العدّ بيبلّش من 0
`a[a.length - 1]` آخر خانة	`a[a.length]` برّا المصفوفة: الكود بيوقع
`new int[n]` الطول ثابت: بعد ما انبنت ما بيتغيّر	`int[] b = a;` مش نسخة: نفس المصفوفة باسمين
`f(a)` دالة بتغيّر خانات a؟ بتتغيّر بالأصل كمان	`cnt[x]++` عدّادات: القيمة نفسها هي الـ index

2D قوانين مصفوفة ثنائية

int[][]

`new int[3][4]` 3 صفوف، 4 عواميد، وكلهن 0	`t[r][c]` صف أوّل، بعدين عامود
`t.length` عدد الصفوف	`t[r].length` عدد العواميد بالصف r
`t[0].length` عدد العواميد بمصفوفة مستطيلة	`t[r]` كل صف هو مصفوفة 1D لحاله
`s[i][i]` القطر الرئيسي، بس بمصفوفة مربّعة	`s[i][n - 1 - i]` القطر الثاني. n هو s.length

for قوانين اللفّات

بـ 1D وبـ 2D

`i < a.length` دايمًا <، مش <=	`max = a[0]` أكبر أو أصغر قيمة: بلّشوا من أوّل خانة، مش من 0
`for r { for c }` على الصفوف: الخارجية r، والداخلية c	`for c { for r }` على العواميد: الخارجية c، والخانة لسّا t[r][c]
`sum = 0` مجموع لكل صف أو عامود؟ صفّروه جوّا اللفّة الخارجية	`count++` عدّ: بيبلّش 0، وبيزيد جوّا if
`where = -1` بحث: إذا ضلّ -1، القيمة مش موجودة	`(double) sum / n` معدّل: (double) قبل القسمة، وإلا بيطلع عدد صحيح

شرح 1D بالصفحات 2-3، شرح 2D بالصفحات 4-5، و6 أسئلة مع الحل بالصفحات 6-9 ←

شرح مصفوفة 1D

1 تعريف وبناء

العدّ بيبلّش من 0

```
int[] a = new int[5];          // 5 cells: all 0
int[] g = {7, 3, 9, 4, 6};
int x = g[2];                  // 9
int last = g[g.length - 1];    // 6
```

0	1	2	3	4
7	3	9	4	6

g[2] = 9، و g[4] = 6

`new int[5]` بتبني 5 خانات، وكلهن بيبلّشوا **0**. أو بتكتبوا القيم مباشرة بين `{ }`. أوّل خانة `g[0]` وآخر خانة `g[g.length - 1]`.

2 لفّة، مجموع ومعدّل

length

```
int sum = 0;
for (int i = 0; i < g.length; i++)
  sum += g[i];
double avg = (double) sum / g.length;
// sum = 29, avg = 5.8
```

خانات: `g.length`	5
المجموع: 6 + 4 + 9 + 3 + 7	29
المعدّل: 29 / 5	5.8

(double) قبل القسمة. بدونها 5 / 29 بيطلع 5، لأنّ قسمة int على int بتقصّ الكسر.

3 أكبر قيمة ووين هي

index + max

```
int max = g[0], pos = 0;
for (int i = 1; i < g.length; i++)
  if (g[i] > max) {
    max = g[i];
    pos = i;
  }
// max = 9 at index 2
```

أكبر قيمة	9
مكانها: pos = 2	2

بنبلّش من **g[0]**، فاللفّة بتبلّش من **i = 1**. وبنحفظ i مع القيمة.

4 عدّ وبحث

لفّة وحدة لإشيين

```
int even = 0, where = -1;
for (int i = 0; i < g.length; i++) {
  if (g[i] % 2 == 0)
    even++;
  if (g[i] == 4)
    where = i;
}
// even = 2 (4 and 6), where = 3
```

كم عدد زوجي: 4 و6	2
وين الـ 4: g[3]	3
إذا الـ 4 مش موجودة	-1

where = -1 بالبداية: إذا ضلّت -1 بعد اللفّة، يعني القيمة مش بالمصفوفة، لأنّ ما في index سالب.

5 قلب المصفوفة

تبديل من الطرفين

```
int[] w = {2, 5, 8, 1, 6};
for (int i = 0; i < w.length / 2; i++) {
  int tmp = w[i];
  w[i] = w[w.length - 1 - i];
  w[w.length - 1 - i] = tmp;
}
// w = {6, 1, 8, 5, 2}
```

6	1	8	5	2

w بعد اللفّة: [6, 1, 8, 5, 2]

بس **w.length / 2** لفّات. إذا بتكمّلوا للآخر، كل زوج بيتبدّل مرّتين وبترجع المصفوفة زي ما كانت.

بالصفحة 3: عدّادات + مصفوفة بدالة + 4 أغلاط بـ 1D ←

6 مصفوفة عدّادات

index = القيمة

```
int[] nums = {12, 7, 22, 35, 17, 42};
int[] cnt = new int[10];
for (int i = 0; i < nums.length; i++)
  cnt[nums[i] % 10]++;
// cnt[2] = 3, cnt[7] = 2, cnt[5] = 1
```

آحاد 2: 12, 22, 42	3
آحاد 7: 7, 17	2
آحاد 5: 35	1

nums[i] % 10 هو رقم الآحاد، وهو نفسه رقم الخانة اللي بنزيد عليها 1.

7 مصفوفة بدالة

نفس المصفوفة

```
public static void doubleAll(int[] arr) {
  for (int i = 0; i < arr.length; i++)
    arr[i] *= 2;
}
```

```
int[] h = {1, 2, 3};
doubleAll(h);
// h = {2, 4, 6}
```

h بعد الدالة {2, 4, 6}

الدالة بتشتغل على **نفس المصفوفة**، مش على نسخة منها. فكل تغيير بـ arr بيبيّن بـ h.

4 أغلاط بـ 1D لازم تنتبهوا إلها

1 => بدل <

✗ غلط: بآخر لفّة i = 5 ، وما في g[5]

```
for (int i = 0; i <= g.length; i++)
```

✓ صح

```
for (int i = 0; i < g.length; i++)
```

آخر خانة هي **length - 1**، لأنّ العدّ بيبلّش من 0.

2 max بيبلّش من 0

✗ غلط: كل القيم سالبة، فـ max بيضل 0

```
int[] neg = {-4, -2, -7};
int max = 0;
```

✓ صح

```
int max = neg[0];
```

الجواب الصح **-2**. والـ 0 مش موجود بالمصفوفة أصلًا.

3 b = a مش نسخة

✗ غلط: copy و g نفس المصفوفة

```
int[] copy = g;
copy[0] = 100;   // g[0] = 100 too
```

✓ صح

```
int[] copy = new int[g.length];
for (int i = 0; i < g.length; i++)
  copy[i] = g[i];
```

نسخة يعني **مصفوفة جديدة**، ولفّة بتنسخ خانة خانة. هيك g[0] بيضل 7.

4 g[g.length] كآخر خانة

✗ غلط: برّا المصفوفة، والكود بيوقع

```
int last = g[g.length];
```

✓ صح

```
int last = g[g.length - 1];
```

طول 5 يعني الخانات من **0 لـ 4**. آخر خانة بـ g هي g[4] = 6.

بالصفحة 4: مصفوفة 2D ، صفوف وعواميد ←

شرح مصفوفة 2D

1 تعريف وبناء

صف أوّل، بعدين عامود

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

`t[1][2]` = صف 1، عامود 2

```
int[][] m = new int[3][4];  // 3 rows, 4 cols: all 0
int[][] t = {{3, 7, 1, 5},
             {2, 8, 4, 6},
             {9, 0, 7, 3}};
int x = t[1][2];            // 4 (row 1, col 2)
```

`new int[3][4]` بتبني 3 صفوف و4 عواميد، وكل الخانات بتبلّش **0**. أو بتكتبوا القيم مباشرة، وكل صف بين `{ }` لحاله.

2 كم صف وكم عامود؟

length

```
int rows = t.length;      // 3
int cols = t[0].length;   // 4
```

t هي مصفوفة من صفوف، فـ `t.length` هو عدد الصفوف، و`t[0].length` هو طول الصف 0.

صفوف: `t.length`	3
عواميد: `t[0].length`	4

جوّا اللفّة استعملوا **t[r].length**: طول الصف اللي انتو فيه.

3 لفّة على الصفوف

مجموع كل صف

```
int[] rowSum = new int[t.length];
for (int r = 0; r < t.length; r++)
  for (int c = 0; c < t[r].length; c++)
    rowSum[r] += t[r][c];
// rowSum = {16, 20, 19}
```

صف 0 — 16	صف 1 — 20	صف 2 — 19

مثلًا صف 0: 3 + 7 + 1 + 5 = 16

الخارجية **r** على الصفوف، والداخلية **c** بتمشي على كل الصف.

4 لفّة على العواميد

مجموع كل عامود

```
for (int c = 0; c < t[0].length; c++) {
  int sum = 0;
  for (int r = 0; r < t.length; r++)
    sum += t[r][c];
  System.out.print(sum + " ");
}
// 14 15 12 14
```

عامود 0: 3+2+9 — 14	عامود 1: 7+8+0 — 15
عامود 2: 1+4+7 — 12	عامود 3: 5+6+3 — 14

هين الخارجية على العواميد. بس الخانة لسّا **t[r][c]**: صف أوّل، بعدين عامود. و`sum = 0` من جديد لكل عامود.

5 كل صف هو مصفوفة 1D

t[r]

```
int[] row = t[1];          // {2, 8, 4, 6}
int rs = 0;
for (int c = 0; c < row.length; c++)
  rs += row[c];
// rs = 20, same as rowSum[1]
```

عدد العواميد : `row.length`	4
مجموع صف 1، زي `rowSum[1]` بكرت 3	20

t[1] هو الصف 1 كمصفوفة 1D، فكل اللي بالصفحات 2-3 بيشتغل عليه. وهاي نفس الخانات، مش نسخة.

بالصفحة 5: القطر + أكبر قيمة + 4 أغلاط بـ 2D ←

6 القطر

بس بمصفوفة مربّعة

```
int[][] s = {{5, 1, 2},
             {3, 6, 4},
             {8, 7, 9}};
int d1 = 0, d2 = 0;
for (int i = 0; i < s.length; i++) {
  d1 += s[i][i];                // 5 + 6 + 9
  d2 += s[i][s.length - 1 - i]; // 2 + 6 + 8
}
// d1 = 20, d2 = 16
```

الرئيسي `s[i][i]`	20
الثاني `s[i][n-1-i]`	16

5	1	2
3	6	4
8	7	9

لفّة **وحدة** بتكفي، لأنّ بالقطر رقم الصف بيحدّد رقم العامود.

7 أكبر قيمة ووين هي

max + مكان

```
int best = t[0][0], br = 0, bc = 0;
for (int r = 0; r < t.length; r++)
  for (int c = 0; c < t[r].length; c++)
    if (t[r][c] > best) {
      best = t[r][c];
      br = r;
      bc = c;
    }
// best = 9 at row 2, col 0
```

أكبر قيمة بـ t (نفس المصفوفة من صفحة 4)	9
مكانها: br = 2 وbc = 0	t[2][0]
كان الأكبر لحد ما وصلنا لصف 2	8

بنبلّش من **t[0][0]** مش من 0، وبنحفظ **r** و**c** مع القيمة.

4 أغلاط بـ 2D لازم تنتبهوا إلها

1 لفّة الصفوف لحد عدد العواميد

✗ غلط: r بيوصل 3، وما في صف 3

```
for (int r = 0; r < t[0].length; r++)
```

✓ صح

```
for (int r = 0; r < t.length; r++)
```

t.length للصفوف، و**t[0].length** للعواميد. بمصفوفة مربّعة الغلط ما بيبيّن، بس بـ 4×3 الكود بيوقع.

2 <= بدل <

✗ غلط: الكود بيوقع بآخر لفّة

```
for (int c = 0; c <= t[r].length; c++)
```

✓ صح

```
for (int c = 0; c < t[r].length; c++)
```

آخر خانة بالصف هي **length - 1**، لأنّ العدّ بيبلّش من 0.

3 المجموع ما رجع لـ 0

✗ غلط: sum برّا اللفّة، فبيتراكم

```
int sum = 0;
for (int c = 0; c < t[0].length; c++) {
  for (int r = 0; r < t.length; r++)
    sum += t[r][c];
  System.out.print(sum + " ");
}
// 14 29 41 55
```

حطّوا **int sum = 0;** جوّا اللفّة الخارجية زي كرت 4، فبيطلع 14 15 12 14 لكل عامود لحاله.

4 [c][r] بدل [r][c]

✗ غلط: t[3] مش موجود، والكود بيوقع

```
sum += t[c][r];
```

✓ صح

```
sum += t[r][c];
```

حتى لو اللفّة الخارجية على العواميد، الخانة دايمًا **[صف][عامود]**.

BagrutAce

3 أسئلة 1D من البنك

1 إزاحة ومجموع موزون

متوسط

أمامكم الفئتان `Shifter` و`Driver` بلغة Java. الدالة `shift` تعدّل المصفوفة المُمرَّرة إليها مباشرة، والدالة `weight` تحسب مجموعًا موزونًا لأوّل `k` عناصر. أجيبوا عن البندين أ-ب.

```
public class Shifter {
  public static void shift(int[] arr) {
    int last = arr[arr.length - 1];
    for (int i = arr.length - 1; i > 0; i--) {
      arr[i] = arr[i - 1];
    }
    arr[0] = last;
  }
  public static int weight(int[] arr, int k) {
    int s = 0;
    for (int i = 0; i < k; i++) {
      s += arr[i] * (i + 1);
    }
    return s;
  }
}
```

```
public class Driver {
  public static void main(String[] args) {
    int[] v = {2, 4, 6, 8};
    Shifter.shift(v);
    System.out.println(v[0] + " " + v[3]);
    System.out.println(Shifter.weight(v, 3));
  }
}
```

أ ما المخرج في السطر الأوّل؟

(أ) 2 8 (ب) 6 8 (ج) 8 6 (د) 8 2

ب ما الرقم المطبوع في السطر الثاني؟

2 نفس المصفوفة أو نسخة؟

متوسط

أمامكم الفئة `Table` والفئة `Tester` بلغة Java. الدالة `make` تنشئ مصفوفة جديدة، والدالة `same` تُعيد المعامل نفسه بدون نسخ. أجيبوا عن البنود أ-جـ.

```
public class Table {
  public static int[] make(int k) {
    int[] arr = {k, k * 2};
    return arr;
  }
  public static int[] same(int[] a) {
    return a;
  }
}
```

```
public class Tester {
  public static void main(String[] args) {
    int[] p = Table.make(3);
    int[] q = Table.same(p);
    int[] r = Table.make(3);
    q[0] = 9;
    System.out.println(p[0] + " " + q[0]
                       + " " + r[0]);
    System.out.println(p == q ? 1 : 0);
    System.out.println(p == r ? 1 : 0);
    System.out.println(p[1] + r[1]);
  }
}
```

أ ما المخرج في السطر الأوّل؟

(أ) 3 9 3 (ب) 9 9 3 (ج) 9 3 3 (د) 3 3 3

ب ما المخرج في السطر الثالث؟

(أ) 1 (ب) 0 (ج) 2 (د) 3

جـ ما المخرج في السطر الرابع؟

3 أطول تسلسل أيّام حارّة

صعب

في دكّان تُحفظ المبيعات اليوميّة بالشواقل في مصفوفة من نمط صحيح `sales`. **يوم حارّ** هو يوم مبيعاته أكبر من 200. نريد طول أطول تسلسل متتالٍ من الأيّام الحارّة. مثلًا من أجل المصفوفة `[150, 250, 310, 180, 220, 240, 260, 100]` الجواب هو 3، لأنّ التسلسل `[250, 310]` طوله 2، والتسلسل `[220, 240, 260]` طوله 3، والأطول هو 3. إذا لم يوجد أيّ يوم حارّ يُعاد 0، فمن أجل `[10, 20, 30]` يُعاد 0، ومن أجل `[250, 260, 270]` يُعاد 3.

أ **اكتبوا عمليّة خارجيّة باسم `longestHotSales` تتلقّى مصفوفة ممتلئة `sales`، وتُعيد طول أطول مقطع متتالٍ كلّ خليّة فيه أكبر من 200. إذا لم توجد خليّة كهذه تُعيد 0. لا تغيّر العمليّة المصفوفة. التوقيع:**
`public static int longestHotSales(int[] sales)`.

ب **ماذا تُعيد العمليّة من أجل المصفوفة `[200, 201, 199, 202, 203]`؟ انتبهوا: الشرط هو أكبر من 200، فالقيمة 200 نفسها ليست حارّة.**

BagrutAce

3 أسئلة 2D من البنك

4 الصف الأكبر مجموعًا

متوسط

أمامكم الفئتان `Grid` و`Driver` بلغة Java. الدالة `rowSum` ساكنة وتجمع عناصر صف في مصفوفة ثنائية، والعملية `main` تبحث عن الصف الأكبر مجموعًا. أجيبوا عن البنود أ-جـ.

```
public class Grid {
  public static int rowSum(int[][] g, int r) {
    int s = 0;
    for (int c = 0; c < g[r].length; c++) {
      s += g[r][c];
    }
    return s;
  }
}
```

```
public class Driver {
  public static void main(String[] args) {
    int[][] g = { {1, 2, 3},
                  {4, 5, 6},
                  {0, 0, 9} };
    int best = -1;
    int bestRow = -1;
    for (int r = 0; r < 3; r++) {
      int t = Grid.rowSum(g, r);
      if (t > best) {
        best = t;
        bestRow = r;
      }
    }
    System.out.println(bestRow + " " + best);
    g[2][0] = 10;
    System.out.println(Grid.rowSum(g, 2));
  }
}
```

أ ما قيمة المتغير `best` بعد انتهاء الحلقة الأولى؟

ب ما المخرج في السطر الأول الذي تطبعه `main`؟

(أ) 2 9 (ب) 0 6 (ج) 1 6 (د) 1 15

جـ ما قيمة `g[2][1] + g[1][0]` بعد السطر `g[2][0] = 10;`؟

5 القطر وتعديل المصفوفة

متوسط

أمامكم الفئة `Grid` والفئة `Tester` بلغة Java. الدالة الساكنة `trace` تجمع القطر الرئيسي لمصفوفة ثنائية، والدالة `bump` تعدّل خانة ثم تُسند إلى معاملها مصفوفة جديدة. أجيبوا عن البندين أ-ب.

```
public class Grid {
  public static int trace(int[][] g) {
    int s = 0;
    for (int i = 0; i < g.length; i++) {
      s = s + g[i][i];
    }
    return s;
  }
  public static void bump(int[][] g, int k) {
    g[0][1] = g[0][1] + k;
    g = new int[1][1];
    g[0][0] = 77;
  }
}
```

```
public class Tester {
  public static void main(String[] args) {
    int[][] m = {{1, 2}, {3, 4}};
    System.out.println(Grid.trace(m));
    Grid.bump(m, 10);
    System.out.println(m[0][1]);
    int[] row = m[1];
    row[0] = 30;
    System.out.println(m[1][0] + Grid.trace(m));
  }
}
```

أ ما المخرج في السطر الثاني؟

ب ما المخرج في السطر الثالث؟

(أ) 35 (ب) 25 (ج) 30 (د) 36

6 دمج نظيف

صعب

شركة نقل تبني رمز دمج من رمزين `s1` و`s2` بأخذ حرف من `s1` ثم حرف من `s2` بالتناوب، ثم إلحاق الباقي من الرمز الأطول في النهاية. مثلًا دمج `ab` و`cd` هو `acbd`، ودمج `ab` و`XYZW` هو `aXbYZW`، ودمج `ab` و`a` هو `aab`. رمز الدمج يُسمّى **نظيفًا** إذا لا يوجد فيه حرفان متجاوران متطابقان. فـ `acbd` نظيف، و`aab` ليس نظيفًا لوجود `aa` في البداية.

أ (1) اكتبوا عمليّة خارجيّة باسم `interleave` تتلقّى نصّين `s1` و`s2`، وتُعيد نصّ الدمج بالتناوب مع إلحاق الباقي.

أ (2) اكتبوا عمليّة خارجيّة باسم `isMergedClean` تتلقّى نصّين `s1` و`s2`، وتُعيد `true` إذا كان ناتج دمجهما نظيفًا، وخلاف ذلك تُعيد `false`. يجب استعمال العمليّة من البند أ (1).

ب اكتبوا عمليّة خارجيّة باسم `countMergedClean` تتلقّى **مصفوفة ثنائيّة الأبعاد** `pairs` من نصوص، كلّ سطر فيها يحتوي بالضبط على نصّين (`pairs[i][0]` و`pairs[i][1]`). تُعيد العمليّة عدد الأسطر التي دمجها نظيف. يجب استعمال العمليّة من البند أ (2). مثلًا إذا كانت `pairs` تحتوي `ab,cd` و`ab,a` فتُعيد 1.

الأسئلة الثلاثة من بنك أسئلة BagrutAce، بنفس صياغة البچروت. حلّوها على ورقة، والحل الكامل بالصفحة 9.

الحل الكامل: 1D

BagrutAce

1 إزاحة ومجموع موزون

step	v
last = arr[3]	8
i = 3	2, 4, 6, 6
i = 2	2, 4, 4, 6
i = 1	2, 2, 4, 6
arr[0] = last	8, 2, 4, 6

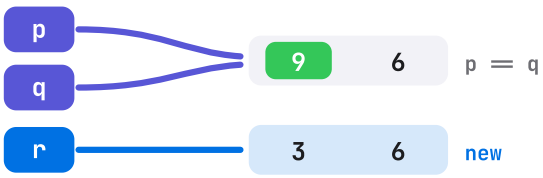
8 6 أ: الجواب (ج). بعد الإزاحة v[0] = 8 و v[3] = 6.

24 ب: 4·3 + 2·2 + 1·8 = 12+4+8.

8, 2, 2, 2 لو اللفّة مشيت من i = 1 لقدّام، الـ 2 كانت بتنسخ حالها لكل الخانات.

الإزاحة لليمين بتمشي من الآخر للأوّل، وبنحفظ last قبل ما ينمحى.

2 نفس المصفوفة أو نسخة؟



9 9 3 أ: الجواب (ب). same بترجّع نفس المرجع، فـ p وq هم نفس المصفوفة، و q[0] = 9 غيّرت p[0] كمان.

1 السطر الثاني: p == q، نفس المرجع.

0 ب: الجواب (ب). == بتقارن مراجع مش قيم: r فيها نفس القيم بس هي مصفوفة تانية.

12 جـ: p[1] + r[1] = 6 + 6.

3 أطول تسلسل أيّام حارّة

i	sales[i]	> 200?	cur	best
0	150	✗	0	0
1	250	✓	1	1
2	310	✓	2	2
3	180	✗	0	2
4	220	✓	1	2
5	240	✓	2	2
6	260	✓	3	3
7	100	✗	0	3

أ longestHotSales

```java
public static int longestHotSales(int[] sales) {
  int best = 0;
  int cur = 0;
  for (int i = 0; i < sales.length; i++) {
    if (sales[i] > 200) {
      cur++;
      if (cur > best)
        best = cur;
    } else {
      cur = 0;
    }
  }
  return best;
}
```

cur طول التسلسل الحالي، وبيرجع 0 بكل يوم مش حارّ. **best** بيحفظ الأطول لهسا. لفّة وحدة بتكفي.

3 المثال الأوّل: أطول تسلسل هو [220, 240, 260].

5 غلط شائع: بدون cur = 0 بالـ else بتعدّوا كل الأيام الحارّة، مش المتتالية بس.

2 ب: 200 مش حارّ، فالتسلسلات هي [201] و[202, 203]، والأطول طوله 2.

الشرط sales[i] > 200 وليس >= 200: هاي النقطة اللي بتوقّع ناس كتير بالبند ب.

الحل الكامل: 2D

BagrutAce

4 الصف الأكبر مجموعًا

r	rowSum(g, r)	t > best?	best	bestRow
0	1+2+3 = 6	6 > -1 ✓	6	0
1	4+5+6 = 15	15 > 6 ✓	15	1
2	0+0+9 = 9	9 > 15 ✗	15	1

15 — أ: الصف 1 هو الأكبر.

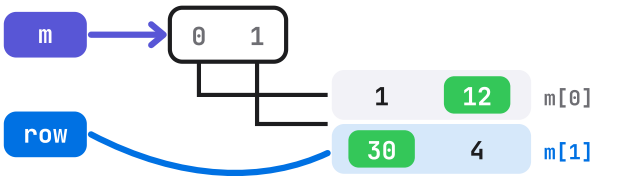
1 15 — ب: الجواب (د). بتنطبع bestRow وبعدها best.

4 — جـ: 0 = g[2][1] و4 = g[1][0]. التغيير صار بـ g[2][0] بس.

19 — والسطر الثاني: 9 + 0 + 10.

شرط t > best بيبدّل بس ونتا نلاقي مجموع أكبر، فـ 9 ما بيغيّر إشي.

5 القطر وتعديل المصفوفة



5 — السطر الأوّل: القطر 4 + 1.

12 — أ: bump بتزيد 10 على g[0][1]، وهاي نفس المصفوفة تبعت main. فـ 2 بيصير 12.

77 ✗ — g = new int[1][1] بتغيّر بس المتغيّر المحلّي g. الـ 77 ما بتوصل لـ m.

35 — ب: الجواب (أ). row هو نفس الصف m[1]، فـ m[1][0] صار 30. القطر لسّا 5، والمجموع 35.

6 دمج نظيف

أ (1) interleave

```
public static String interleave(
    String s1, String s2) {
  String res = "";
  int i = 0;
  while (i < s1.length() || i < s2.length()) {
    if (i < s1.length())
      res += s1.charAt(i);
    if (i < s2.length())
      res += s2.charAt(i);
    i++;
  }
  return res;
}
```

اللفّة بتكمّل **طول ما في حرف** بواحد من النصّين، فالباقي من الأطول بينضاف لحاله. مثال interleave("ab", "XYZW"):

i	s1.charAt(i)	s2.charAt(i)	res
0	a	X	aX
1	b	Y	aXbY
2	-	Z	aXbYZ
3	-	W	aXbYZW

acbd — السطر ab, cd: ما في حرفين متجاورين متطابقين، يعني نظيف.

aab — السطر ab, a: aa بالبداية، يعني مش نظيف. فالجواب للمثال 1

أ (2) isMergedClean

```
public static boolean isMergedClean(
    String s1, String s2) {
  String m = interleave(s1, s2);
  for (int i = 0; i < m.length() - 1; i++)
    if (m.charAt(i) == m.charAt(i + 1))
      return false;
  return true;
}
```

ب countMergedClean

```
public static int countMergedClean(
    String[][] pairs) {
  int count = 0;
  for (int i = 0; i < pairs.length; i++)
    if (isMergedClean(pairs[i][0],
                      pairs[i][1]))
      count++;
  return count;
}
```

pairs[i][0] و pairs[i][1]: لفّة وحدة على الصفوف بتكفي، لأنّ كل صف فيه بالزبط نصّين. والبند ب بينادي أ (2)، وأ (2) بينادي أ (1).

بدكم تتدرّبوا على مصفوفات؟

حلّوا على BagrutAce وAce بيصلّحلكم على طول.

bagrutace.com ← يلّا نبلّش