Question 1

Not answered

Mark 0.00 out of 10.00

[Unique]

Viết chương trình nhập vào một dãy số và đếm số lượng phần tử khác nhau trong dãy số đấy.

Gợi ý: Sử dụng hàm unique trong thư viện algorithm.

Đầu vào

Đầu vào từ bàn phím gồm 2 dòng.

- Dòng đầu tiên chứa số nguyên n là số lượng phần tử của dãy số.
- Dòng tiếp theo chứa n số nguyên tương ứng với các phần tử trong dãy $(n \leq 10^5)$.

Đầu ra

In ra màn hình số lượng phần tử khác nhau trong dãy số đấy.

Answer: (penalty regime: 0 %)

```
1
```

Question 2

Not answered

Mark 0.00 out of 10.00

[next_permutation]

Cho một dãy số gồm n chữ số là một hoán vị của dãy $1, 2, \ldots, n$. Viết chương trình in ra hoán vị ở vị trí tiếp theo (danh sách các hoán vị được sắp xếp theo thứ tự từ điển) của hoán vị đã cho.

Đầu vào

Đầu vào từ bàn phím gồm 2 dòng:

- Dòng đầu tiên chứa số nguyên n
- Dòng tiếp theo chứa n số nguyên là một hoán vị của dãy $1, 2, \ldots, n$ $(n \leq 10^5)$

Đầu ra

In ra màn hình hoán vị tiếp theo của hoán vị đã cho.

Gợi ý: Dùng hàm `next_permutation (trong thư viện algorithm)`

For example:

Input	Result
3 2 3 1	3 1 2

Answer: (penalty regime: 0 %)

```
1
```

▸ SHOW/HIDE QUESTION AUTHOR'S SOLUTION (CPP)

Question 3

Not answered

Mark 0.00 out of 10.00

[InputFunction]

Cho nguyên mẫu hàm sau để nhập vào một số nguyên từ bàn phím: `void input(int &);`

Các bạn hãy viết định nghĩa hàm trên (chỉ cần định nghĩa hàm này)

Answer: (penalty regime: 0 %)

```
1
```

Question 4

Not answered

Mark 0.00 out of 10.00

[PerfectNumberFunction]

Cho nguyên mẫu hàm sau để kiểm tra một số có phải là số hoàn thiện không: `bool isPerfect(int);`

Hàm `isPerfect`👎 sẽ trả về giá trị **true** nếu n là số hoàn thiện và giá trị **false** trong trường hợp ngược lại. Các bạn hãy viết định nghĩa hàm trên (chỉ cần định nghĩa hàm này)

Định nghĩa: số hoàn thiện là số có tổng các ước số (không kể chính nó) bằng chính số đó.

Answer: (penalty regime: 0 %)

```
1
```

Question 5

Not answered

Mark 0.00 out of 10.00

[CanPlaceFlowers]

Giả sử bạn là nhân viên của một công ty cây xanh, được giao cho chăm sóc một luống đất. Luống đất đó được chia thành nhiều ô đất nhỏ có kích cỡ bằng nhau.

Trước khi được giao cho bạn chăm sóc, trên luống hoa đã được trồng sẵn một số cây xanh tán rộng.

Một ngày đầu xuân, công ty gửi đến cho bạn một số cây hoa và yêu cầu bạn trồng vào những vị trí còn trống trong luống hoa.

Đây là những cây hoa hướng sáng và chúng không thể sống được dưới bóng râm của cây khác, đặc biệt là cây tán rộng. Vì vậy, bạn không thể trồng chúng tại ô bên cạnh những ô đã được trồng cây xanh tán rộng.

Và nếu trồng những cây hoa bên cạnh nhau, chúng sẽ tranh giành phân bón và nước và sau cùng, cả hai cây sẽ cùng chết.

Viết hàm `bool canPlaceFlowers (int flowerbed[], int n, int k)` kiểm tra xem có thể trồng hết số hoa mới được cấp vào trong luống đất của bạn hay không.

Hàm nhận đầu vào là mảng $flowerbed$ ứng với luống đất có n ô đất, đánh dấu những vị trí trong luống đất đã được trồng cây xanh trước đó (giá trị 1 ứng với vị trí cây đã được trồng và 0 ứng với vị trí đất trống) và số lượng hoa được phát thêm k. Hàm trả về $true$ nếu có thể trồng hết toàn bộ k cây hoa vào luống đất và $false$ trong trường hợp ngược lại.

For example:

Test	Input	Result
`canPlaceFlowers(flowerbed, n, k)`	`10 1` `0 0 0 1 1 1 1 1 0 1`	`true`

Answer: (penalty regime: 0 %)

```
1
```

Question 6

Not answered

Mark 0.00 out of 10.00

[HammingDistance]

Khoảng cách **Hamming** giữa hai số nguyên là số lượng vị trí khác nhau giữa hai dãy bits tương ứng của chúng. Ví dụ: Khoảng cách Hamming giữa **1 (001)** và **4 (100)** là 2.

Cho hai số nguyên `x` và `y`, hãy viết hàm `int hammingDistance(int x, int y)` trả về khoảng cách Hamming giữa hai số này.

Khoảng cách Hamming là cái tên được đặt theo tên của Richard Hamming, người giới thiệu lý thuyết này trong tài liệu có tính cơ sở của ông về *mã phát hiện lỗi và sửa lỗi* (*error-detecting and error-correcting codes*). Nó được sử dụng trong kỹ thuật viễn thông để tính số lượng các bit trong một từ nhị phân (*binary word*) bị đổi ngược, như một hình thức để ước tính số lỗi xảy ra trong quá trình truyền thông, và vì thế, đôi khi, nó còn được gọi là **khoảng cách tín hiệu** (*signal distance*). Việc phân tích trọng số Hamming của các bit còn được sử dụng trong một số ngành, bao gồm lý thuyết tin học, lý thuyết mã hóa, và mật mã học. Tuy vậy, khi so sánh các dãy ký tự có chiều dài khác nhau, hay các dãy ký tự có xu hướng không chỉ bị thay thế không thôi, mà còn bị ảnh hưởng bởi dữ liệu bị lồng thêm vào, hoặc bị xóa đi, phương pháp đo lường phức tạp hơn, như khoảng cách Levenshtein (*Levenshtein distance*) là một phương pháp có tác dụng và thích hợp hơn.

For example:

Test	Input	Result

`int x, y;`
`cin >> x >> y;`
`cout << hammingDistance(x, y);`
	`1 4`	`2`

Answer: (penalty regime: 0 %)

```
1
```

Question 7

Not answered

Mark 0.00 out of 10.00

[Harmonic]

Trong toán học, số hài hòa thứ n là tổng nghịch đảo của n số tự nhiên đầu tiên:

$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}$

Số hài hòa đã được nghiên cứu từ thời cổ đại và đóng vai trò quan trọng trong các ngành khác nhau của lý thuyết số. Đôi khi chúng còn được gọi là chuỗi hài hòa, liên quan chặt chẽ đến hàm Riemann zeta, và xuất hiện trong các biểu thức của nhiều hàm đặc biệt.

Các số hài hòa gần xấp xỉ với hàm logarithm tự nhiên. Năm 1737, Leonhard Euler dùng sự phân kỳ của chuỗi hài hòa để chứng minh là có vô số các số nguyên tố. Công trình của ông sau đó được mở rộng bởi Bernhard Riemann vào năm 1859, dẫn đến giả thuyết nổi tiếng của Riemann về sự phân bố của các số nguyên tố.

Hãy viết hàm `double harmonic(int n)` để tính số hài hào thứ n.

Lưu ý: chỉ cần viết hàm như đề bài yêu cầu, không cần viết hàm `main()`, không cần viết các câu lệnh `#include, using namespace std ...`

For example:

Test	Input	Result

`int n;`
`cin >> n;`
`cout << fixed << setprecision(2) << harmonic(n);`
	2	1.50

Answer: (penalty regime: 0 %)

```
1
```

Question 8

Not answered

Mark 0.00 out of 10.00

[PasswordEntry]

Với kiểu nhập mật khẩu truyền thống, bạn dễ bị nhìn trộm, tức là kẻ xấu theo dõi bạn lúc nhập mật khẩu hay mã PIN và dùng nó để truy cập vào tài khoản của bạn. Một cách để đối phó với vấn đề này là sử dụng một hệ thống sinh thử thách đáp ứng ngẫu nhiên. Trong những hệ thống này, bạn sẽ nhập những thông tin khác nhau dựa trên thông tin bí mật bạn giữ để đáp ứng một thử thách ngẫu nhiên được sinh ra. Thử xem xét trường hợp một mật khẩu là một mã PIN 5 chữ số (00000 đến 99999). Mỗi chữ số được gán một số ngẫu nhiên là 1, 2, hoặc 3. Bạn sẽ nhập những con số ngẫu nhiên tương ứng với mã PIN của bạn thay vì chính mã PIN của bạn.

Ví dụ, xét trường hợp mã PIN của bạn là 12345. Để xác thực quyền truy cập, bạn sẽ được hiển thị thông tin như sau:

```
PIN: 0 1 2 3 4 5 6 7 8 9
NUM: 3 2 3 1 1 3 2 2 1 3
```

Bạn sẽ nhập 23113 thay vì 12345. Điều này không làm lộ mật khẩu của bạn kể cả khi kẻ xấu biết được bạn nhập gì vì 23113 có thể đại diện cho các mã PIN khác, như 69440 hay 70439. Lần kế tiếp bạn xác thực, một dãy số ngẫu nhiên sẽ được sinh ra, chẳng hạn:

```
PIN: 0 1 2 3 4 5 6 7 8 9
NUM: 1 1 2 3 1 2 2 3 3 3
```

Hãy viết hàm `vector<int> getCorrectResponse(vector<int> pin, vector<int> num)` trả về một chuỗi đáp ứng tương ứng với mã PIN và dãy số sinh ngẫu nhiên cho từng chữ số. `pin` là một vector chứa các chữ số của mã PIN, `num` là một vector chứa các kí tự sinh ngẫu nhiên tương ứng với các chữ số từ 0 đến 9.

Viết hàm `vector<int> getDigits(unsigned int number)` nhận vào một số nguyên và trả về mảng chứa các chữ số của số nguyên đó.

Viết hàm `void printDigits(vector<int> digits)` nhận đầu vào là một mảng chứa các chữ số của một số và in ra số đó (in các chữ số liền nhau).

Bạn có thể giả sử người dùng không thích dùng số 0 ở đầu mật khẩu, nên mật khẩu sẽ không có số 0 ở đầu.

Lưu ý: chỉ cần viết các hàm như đề bài yêu cầu, không cần viết hàm `main()`, không cần viết các câu lệnh `#include, using namespace std ...`

For example:

Test	Input	Result

`unsigned int pin, num;`
`cin >> pin >> num;`
`vector<int> digitsOfPin = getDigits(pin);`
`vector<int> digitsOfNum = getDigits(num);`
`vector<int> correctResponse = getCorrectResponse(digitsOfPin, digitsOfNum);`
`printDigits(correctResponse);`
	12345
	3231132213
		23113

Answer: (penalty regime: 0 %)

```
1
```

Question 9

Not answered

Mark 0.00 out of 10.00

[MEDIAN - HARD]

Viết hàm `int Median(int a[], int n)`, tìm ra số trung vị (số lớn thứ ba) trong dãy 5 số nguyên mà không dùng quá 6 phép so sánh. KHÔNG DÙNG BẤT CỨ PHƯƠNG PHÁP SẮP XẾP NÀO.

Chú ý: Bạn chỉ phải viết hàm theo yêu cầu, hàm `main()` và thư viện đã có sẵn.

For example:

Input	Result
1 2 3 4 5	3

Answer: (penalty regime: 0 %)

```
1
```

▸ **SHOW/HIDE QUESTION AUTHOR'S SOLUTION (CPP)**

Question 10

Not answered

Mark 0.00 out of 10.00

[WinterIsComing]

Mùa đông đang đến! Khi tuyết rơi, gió thổi, con sói đơn độc sẽ chết nhưng đàn sói sẽ sống.

Thực ra thì nếu bạn thiết kế một hệ thống sưởi ấm với bán kính cố định hợp lý, sưởi ấm mọi ngôi nhà thì không có ai phải chết rét cả. Bạn sẽ được nhận vị trí của các ngôi nhà và lò sưởi trên một đường thẳng, hãy tìm bán kính nhỏ nhất của các lò sưởi để toàn bộ các nhà được sưởi ấm bởi chúng.

Viết hàm `int findRadius(vector<int> houses, vector<int> heaters)` nhận đầu vào là hai mảng, mảng `houses` chứa vị trí của các ngôi nhà, mảng `heaters` chứa vị trí của các lò sưởi, hàm trả về bán kính tối thiểu của các lò sưởi để sưởi ấm toàn bộ các nhà.

Lưu ý:

1. Số lượng nhà và lò sưởi là một số không âm và không vượt quá 25000.
2. Vị trí của các ngôi nhà và lò sưởi là một số không âm và không vượt quá 10^9.
3. Miễn là ngôi nhà ở trong bán kính của lò sưởi, nó sẽ được sưởi ấm.
4. Mọi lò sưởi đều tuân theo một bán kính chuẩn và bán kính sưởi ấm là như nhau.

For example:

Input	Result
3 1 1 2 3 2	1
4 2 1 2 3 4 1 4	1
2 1 1 5 2	3

Answer: (penalty regime: 0 %)

```
1
```

Back to Course