●●●● Very Frequent 0/20

Basic Complete Search

Authors: Darren Yao, Dong Liu
Contributors: Brad Ma, Nathan Gong

Problems involving iterating through the entire solution space.

Language: Python Edit This Page

TABLE OF CONTENTS

RESOURCES

IUSACO	**6 - Complete Search**	module is based off this

In many problems (especially in Bronze) it suffices to check all possible cases in the solution space, whether it be all elements, all pairs of elements, or all subsets, or all permutations. Unsurprisingly, this is called **complete search** (or **brute force**), because it completely searches the entire solution space.

Maximum Distance

CF - Easy

Focus Problem – try your best to solve this problem before continuing!

Solution - Maximum Distance

We can iterate through every pair of points and find the square of the distance between them, by squaring the formula for Euclidean distance:

$$\text{distance}[(x_1, y_1), (x_2, y_2)]^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2.$$

Maintain the current maximum square distance in `max_squared`.

Warning!

Make sure to submit with PyPy, as this solution TLEs using normal Python.

Copy PYTHON

```python
n = int(input())
x = list(map(int, input().split()))
y = list(map(int, input().split()))

max_squared = 0  # stores the current maximum
for i in range(n):  # for each first point
    for j in range(i + 1, n):  # and each second point
```

A couple notes:

- Since we're iterating through all pairs of points, we start the j loop from $j = i + 1$ so that point i and point j are never the same point. Furthermore, it makes it so that each pair is only counted once. In this problem, it doesn't matter whether we double-count pairs or whether we allow i and j to be the same point, but in other problems where we're counting something rather than looking at the maximum, it's important to be careful that we don't overcount.
- Secondly, the problem asks for the square of the maximum Euclidean distance between any two points. Some students may be tempted to maintain the maximum distance in an integer variable, and then square it at the end when outputting. However, the problem here is that while the square of the distance between two integer points is always an integer, the distance itself isn't guaranteed to be an integer. Thus, we'll end up shoving a non-integer value into an integer variable, which truncates the decimal part.

The following solution correctly stores the maximum distance in a floating point variable.

Copy PYTHON

```python
import math

n = int(input())
x = list(map(int, input().split()))
y = list(map(int, input().split()))

max_dist = 0
for i in range(n):
    for j in range(i + 1, n):
        dx = x[i] - x[j]
        dy = y[i] - y[j]
        square = dx * dx + dy * dy
        max_dist = max(max_dist, math.sqrt(square))

print(int(max_dist**2))
```

However, it still fails on the following test case (it outputs 12, while the correct answer is 13):

```
2
0 3
2 0
```

Rounding suffices (`round(MaxDistance ** 2)`), but the takeaway is that you should stick with integers whenever possible.

Problems