BaekJoon Code Algorithm Solution Collection

Park Cheon Bok

Q 1000. A+B

Two Input Values A and B. Return A+B

|  |
| --- |
| import sys  A, B = map(int, sys.stdin.readline().split())  print(A+B) |

★ sys.stdin.readline() = If Q needs multi line input testcases, Using “input()” will result in a verdict that will take longer than using “sys.stdin.readline()”.

Q 1001. A-B

Two Input Values A and B. Return A-B

|  |
| --- |
| import sys  A, B = map(int, sys.stdin.readline().split())  print(A-B) |

Q 1002. Turret

Two Turret staff, A and B. Each A and B calculated distance from their Turret Area to the enemy.

A’s position is (x1, x2). B’s position is (x2, y2).

The distance between A to the enemy is r1.

The distance between B to the enemy is r2.

Where can the enemy be located?

1. Testcase count = T

2. x1, y1, r1, x2, y2, r2 ( -10,000 <= x1, y1, x2, y2 <= 10,000), (10,000 >= r1, r2)

3. If the enemy can be anywhere, return -1

Testcase Sample

|  |  |
| --- | --- |
| 3  0 0 13 40 0 37  0 0 3 0 7 4  1 1 1 1 1 5 | 2  1  0 |

The solution link : <https://zifmfmphantom.tistory.com/107>

A and B described the enemy and Turret distance as r1, r2.

That means, the enemy is within the perimeter of circle with radii r1, r2.

This problem is solved by using “**Inscribed and circumscribed circle**”.

“d” is distance between A Point to B Point.

“d” equation =

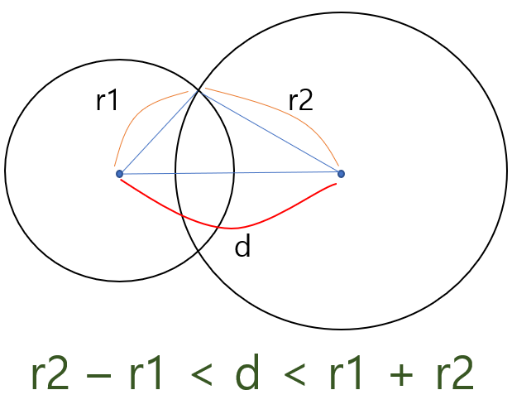
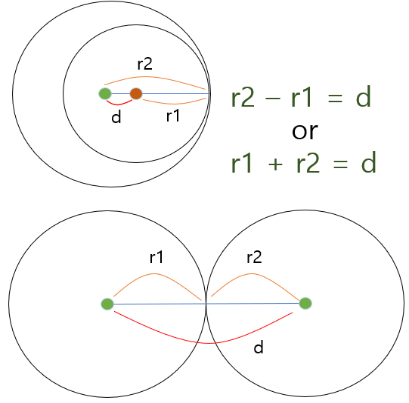
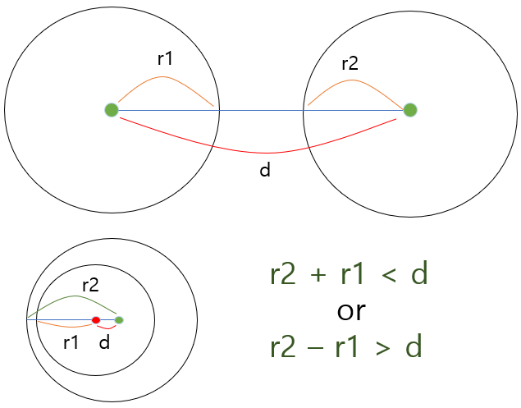
 Figure . Outer Product Figure . Inner Product Figure 3. Two Points overlap

Figure 4. “d = 0” means Two Turrets are in the same location.

That time, “r1 = r2” means Two circles are same.

|  |
| --- |
| import sys  from math import sqrt *# sqrt : 제곱근 함수*  inputF = sys.stdin.readline *# 입력 함수 텍스트 간소화*  def ***TestFunction***(TList : list):  *# d = 두 점의 거리 (Distance)*  d = sqrt((TList[**0**] - TList[**3**])\*\***2** + (TList[**1**] - TList[**4**])\*\***2**)  '''  경우 1 : 동심원 (두 점의 위치가 같고(d=0) 거리(r1,r2)도 서로 같다. => 무한대(-1)  경우 2 : 만나지 않음(0) = d가 두 반지름보다 길다 = 두 원이 떨어짐.  d가 두 반지름의 차보다 작다 = 하나의 원은 다른 원 안에 존재.  경우 3 : 1개의 점에서 만남(1) = d가 두 반지름의 거리 합과 같다. (외접)  d가 두 반지름의 거리 차와 같다. (내접)  경우 4 : 일반적인 경우(2) = 위의 특이 경우에 해당하지 않으면 두 점과 만남.  '''  if d == **0** and TList[**2**] == TList[**5**]: *# Figure 4*  return -**1**  elif d > TList[**2**]+TList[**5**] or d < abs(TList[**2**]-TList[**5**]): *# Figure 1*  return **0**  elif d == TList[**2**]+TList[**5**] or d == abs(TList[**2**]-TList[**5**]): *# Figure 2*  return **1**  else: *# Figure 3*  return **2**  if **\_\_name\_\_** == '\_\_main\_\_':  T = int(inputF()) *# TestCase Count*  for \_ in range(T):  TestList = list(map(int, inputF().split()))  print(TestFunction(TestList)) *# 구현 함수* |

Q 1003. 피보나치 함수

This is a C++ Function to describe the “Fibonacci Function”.

|  |
| --- |
| int fibonacci(int n) {  if (n == **0**) {  printf("0");  return **0**;  } else if (n == **1**) {  printf("1");  return **1**;  } else {  return fibonacci(n‐**1**) + fibonacci(n‐**2**);  }  } |

If “Fibonacci(3)” executed, Number 1 and 0 are called each twice and once.

If “Fibonacci(N)” executed. How many times will be called Number 1 and 0?

1. Testcase T = count. First Input

2. N ( N= 0 or N <= 40)

Testcase Sample

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Output | Input | Output |
| 3  0  1  3 | 1 0  0 1  1 2 | 2  6  22 | 5 8  10946 17711 |

★ First Solution Used “Recursive Function” likes Sample C++ codes.

But the result is “Time Over”. Q does not want to solve using recursive.

-> Using Recursive Function (Time Over)

|  |
| --- |
| Cnt1 = **0**  Cnt0 = **0**  def ***FiboCnt***(N : int):  global Cnt1, Cnt0  if N == **0**:  Cnt0 += **1**  elif N == **1**:  Cnt1 += **1**  else:  FiboCnt(N-**1**)  FiboCnt(N-**2**) |

The Key of the solution is finding the rules of recursive.

That means, “the next value is equal to the sum of the previous two values”.

ex. N=3 -> 2+1 = (1+0) + 1

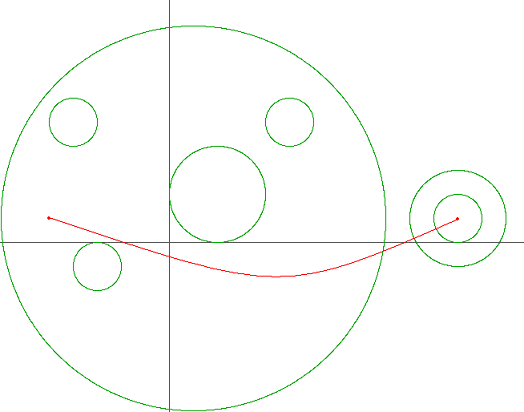
N=4 -> 3+2 = (1+0+1) + (1+0)

N=5 -> 4+3 = (1+0+1+1+0) + (1+0+1)

|  |
| --- |
| '''3이면 2+1 (1+0) + 1  4이면 3+2 (1+0+1) + (1+0)  5이면 4+3 (1+0+1+1+0) + (1+0+1)  입력 수의 전 수와 전전수를 더한다.  이는 0과 1의 개수도 전 수와 전전 수의 값에 영향을 받는다는 의미. '''  import sys  inputF = sys.stdin.readline  T = int(inputF())  Cnt0 = [**1**, **0**]  Cnt1 = [**0**, **1**]  *# 전 단계의 0, 1 출현 개수 + 전전 단계의 0, 1 출현 개수*  def ***ZeroOneCount***(N : int):  for i in range(**2**, N+**1**): *# 0과 1은 직접 처리함*  Cnt0.append(Cnt0[i-**1**]+Cnt0[i-**2**])  Cnt1.append(Cnt1[i-**1**]+Cnt1[i-**2**])  for \_ in range(T):  N = int(inputF())  if N == **0**: *# 0 직접 처리*  print("1 0")  elif N == **1**: *# 1 직접 처리*  print("0 1")  else:  ZeroOneCount(N)  print(f"{Cnt0.pop()} {Cnt1.pop()}") *# 최종 계산 결과는 마지막 인덱스에 있다. (pop)*  *# 1회분 끝났으면 초기화*  Cnt0 = [**1**, **0**]  Cnt1 = [**0**, **1**]  *# 다이나믹 프로그래밍* |

Q 1004 어린왕자

The spaceship of the little prince must avoid as much as possible the planets to reach the destination.

 Milky Way map

The red line is the route that avoids planets as much as possible to the destination. (ex. Minimum 3)

1. Testcase T. 2. From the next line, the departure (x1, y1) and the destination (x2, y2)

3. The number of planets like ‘n’ 4. Each line, the midpoint and radius of the planet (cx, cy, r)

Testcase Sample : <https://www.acmicpc.net/problem/1004>

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명텍스트, 사람이(가) 표시된 사진

자동 생성된 설명

The solution is “The distance between two points”.

If a planet contains a departure or destination,

The distance between departure and planet or destination and planet is less than r.

|  |
| --- |
| import sys  from math import sqrt *# 제곱근 함수, "\*\* 0.5"로 계산해도 동일 결과*  inputF = sys.stdin.readline  T = int(inputF())  for \_ in range(T):  x1, y1, x2, y2 = map(int, inputF().split())  pT = int(inputF())  result = **0** *# 통과하는 행성 수*  for \_ in range(pT):  cx, cy, r = map(int, inputF().split())  *# 출발지, 목적지가 같은 행성 안에 포함되는지 파악. (해당 경우는 1 더할 필요가 없기 때문)*  if sqrt((x1-cx)\*\***2** + (y1-cy)\*\***2**) < r:  if sqrt((x2-cx)\*\***2** + (y2-cy)\*\***2**) < r:  pass  else:  *# 출발지, 목적지가 같은 행성 내에 없으면*  result += **1**  elif sqrt((x2-cx)\*\***2** + (y2-cy)\*\***2**) < r:  result += **1**  print(result) |
| '''  XOR은 두 비교가 다른 값일 때 (1, 0) 또는 (0, 1) True를 반환한다.  조건에서 보면 출발지, 도착지 두 점이 모두 같은 행성 안에 있으면 (1, 1) 오히려 반영하지 않는다.  둘 중 하나의 점만 행성 안에 포함되어 있어야 반영하는 구조인 것이다. (1,0), (0,1)일 때.  그래서 XOR을 사용하며 이를 통해 조건문 하나를 없앨 수 있다.  '''  A = **1** if sqrt((x1-cx)\*\***2** + (y1-cy)\*\***2**) < r else **0**  B = **1** if sqrt((x2-cx)\*\***2** + (y2-cy)\*\***2**) < r else **0**  if A^B==**1**: result+=**1** *# XOR 연산으로 행성 통과 유무 계산* |