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 =

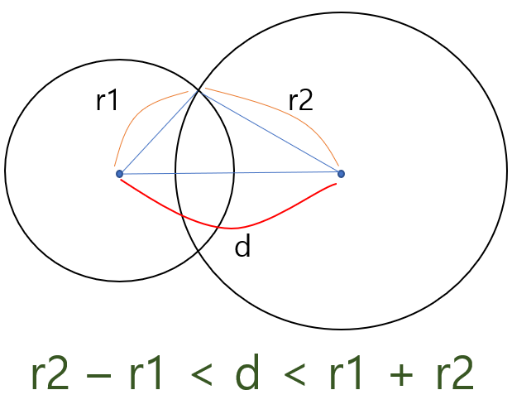
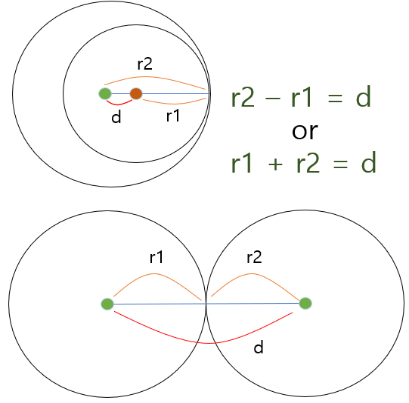
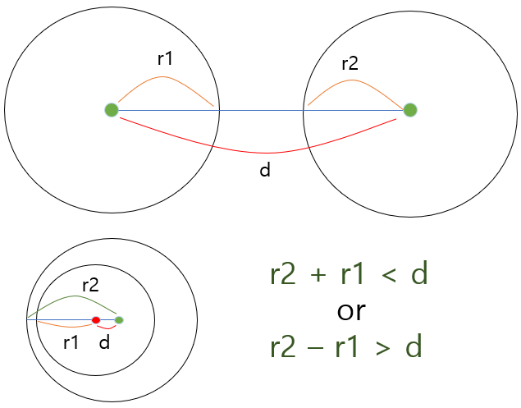
 Figure 1. Outer Product Figure 2. Inner Product Figure 3. Two Points overlap

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

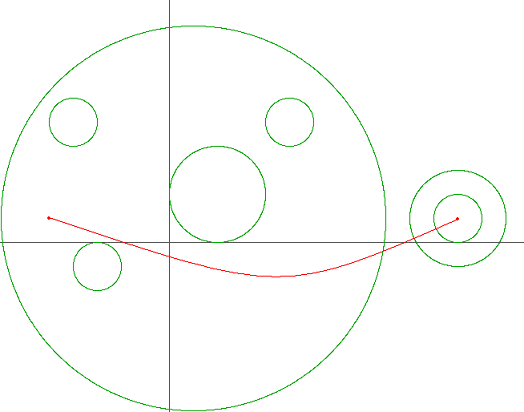
N=4 -> 3+2 = (1+0+1) + (1+0)

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

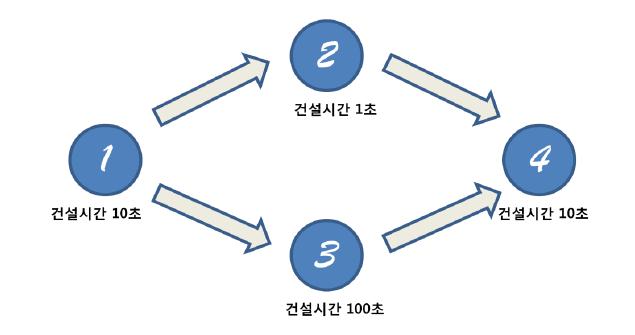
The solution is “The distance between two points”.

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* |

Q 1005. ACM Craft



If the building of #1 is constructed, building of #2 and #3 can begin. (Can start same time)

And, in order to build the #4, #1 and #3 must be constructed. “Result output is 120s”

Link : <https://www.acmicpc.net/problem/1005>

To solve the problem, need to use the algorithm of “Topological alignment”.

Study link : <https://m.blog.naver.com/ndb796/221236874984>

|  |
| --- |
| import sys  from collections import deque *# Queue 함수를 사용하기 위한 데큐*  inputF = sys.stdin.readline  T = int(inputF())  Build\_Time = dict() *# 건설 시간 저장 딕셔너리*  tmp = **0** *# 건설 시간 저장하는 임시 변수*  def ***Topology\_Sort***(): *# 위상정렬 알고리즘*  q = deque() *# Deque 객체*  '''진입 차수가 0인 건물을 찾는다.  처음 짓기 시작한 건물은 첫 입력에 진입 차수가 0일 것이다.  건물을 짓기 위해 소요되는 시간은 'time' 리스트에 누적한다.  그리고, q에 해당 건물 번호를 추가한다.'''  for i in range(**1**, N+**1**): *# 진입 차수가 0인 건물을 찾아서 건설 시간 저장*  if indegree[i] == **0**:  time[i] = Build\_Time[i]  q.append(i)  '''q에 데이터가 있다는 것은 아직 목표 건물까지 도달하지 않았음을 의미.  목표 건물까지 도달하는 과정을 q를 통해 조절'''  while q:  value = q.popleft() *# 첫 1 추출*  tmp = time[value] *# 추출된 건물에 대한 건설 시간*  for i in Rule[value]: *# 해당 건물 다음으로 건설하는 건물 차수 1 빼기.*  *# 진입 차수가 2개 이상인 경우는 이전 값과 비교해 큰 값으로 대체.*  time[i] = max(time[i], tmp + Build\_Time[i]) *# 해당 건물까지의 건설 최대 소요 시간.*  indegree[i] -= **1** *# 진입 차수 1 차감.*  if indegree[i] == **0**: *# 집입 차수가 0이 되면 추가.*  q.append(i)  for \_ in range(T):  N, K = map(int, inputF().split()) *# 건물 개수 N, 규칙 개수 K*  Values = list(map(int, inputF().split())) *# 건물 건설 시간 리스트*  indegree = [**0**] \* (N+**1**) *# 각 건물의 진입 차수 리스트*  time = [**0**] \* (N+**1**) *# 각 건물의 건설 시간 (추후 누적)*  Rule = [[] for \_ in range(N+**1**)] *# 건물 규칙 정보 저장 리스트*  for i in range(**1**, N+**1**): *# 각 건물에 대한 시간 딕셔너리*  Build\_Time[i] = Values[i-**1**]  for i in range(K): *# 간선 규칙 정보 담기*  outV, inV = map(int, inputF().split())  *# 뻗어 나가는 가지가 2개 이상일 경우 처리 방법*  Rule[outV].append(inV) *# 좌측(outV 완료한 건물) 우측(inV 다음 건물)*  *# 들어온 가지 수는 곧 진입 차수를 의미.*  indegree[inV] += **1** *# 진입 차수 1 증가*  W = int(inputF())  Topology\_Sort()  print(time[W]) |

Q 1008. A/B

Two Values A and B in input, return A/B.

★ Operator ‘/’ returns the quotient with the prime number.

Operator ‘//’ returns the only integer quotient.

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

Q 1009. 분산처리

The A has 10 Computers. They are processing each data. -> 1~10th data processed by 1~10th computer,

11~Nth data processed by 1~10th computer.

Total number of data is . (ex. 1 6 -> = 1)

For each test case, the program returns the computer number which processes last data.

Testcase Sample

|  |  |  |
| --- | --- | --- |
| Input | Output | Process |
| 2  1 6  6 2 | 1  6 | 1 6 = 1^6 = 1. Computer = 1  6 2 = 6^2 = 36 Computer = 6 |

Using “Exponential Calculation” is “Time Out”. So, find out the rule of solution.

There are 10 computers. So, divide by 10 for the answer.

The remainder has the rule. Finding rules can reduce computation.

테이블이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| **import** **sys**  inputF = sys.stdin.readline  N = int(inputF())  **for** i **in** range(N):  a, b = map(int, inputF().split())  rules = []  n = 1  **while** **True**: *# 나머지 규칙 찾아냄 (계산 과정 줄이기)*  **if** a\*\*n%10 **in** rules:  **break**  **else**:  rules.append(a\*\*n%10) *# 나머지 규칙 값을 저장하는 리스트*  n += 1  b = b%len(rules) *# 나머지 규칙 개수로 지수 계산 삭제*  **if** rules[b-1] == 0: *# 나머지 0이 나오는 경우 = 10 번째 PC*  print('10')  **else**:  print(rules[b-1]) |

Q 1010. 다리 놓기

The west has N sites, the east has M sites. ( N <= M )

재원 wants to connect the bridges. The one site can be connected at a one site.

The number of bridges connected to each other is N. (Not duplication!)

Find the number of cases which connect the bridges if possible.

Testcase Sample

|  |  |
| --- | --- |
| Input | Output |
| 3  2 2  1 5  13 29 | 1  5  67863915 |

★ Have to use “**Combination**” of mathematics.

Study link : <https://ourcalc.com/%EC%A1%B0%ED%95%A9-%EA%B3%84%EC%82%B0%EA%B8%B0/>

텍스트이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| **import** **sys**  **from** **math** **import** factorial *# 팩토리얼 계산 함수*  inputF = sys.stdin.readline  T = int(inputF())  **for** \_ **in** range(T):  N, M = map(int, inputF().split())  *''' 조합 계산식 변형. nCr = nPr / r! '''*  mPn = factorial(M) // factorial(M-N)  nFacto = factorial(N)  print(mPn//nFacto) |

Q 1011. Fly me to the Alpha Centauri

텍스트이(가) 표시된 사진

자동 생성된 설명

If the spaceship has moved distance k, can move k-1, k, k+1 next time.

(ex. If k=1, next time can move 0, 1, 2)

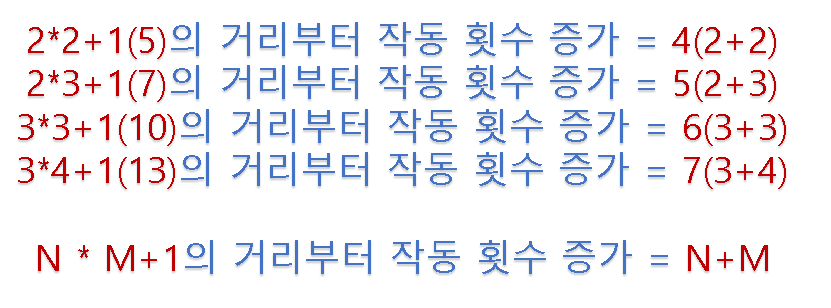
But, First and Last time should move only 1 distance.

How can reach y at minimum movement from x?

Solution Link = <https://zifmfmphantom.tistory.com/14>

There are rules of solution.

테이블이(가) 표시된 사진

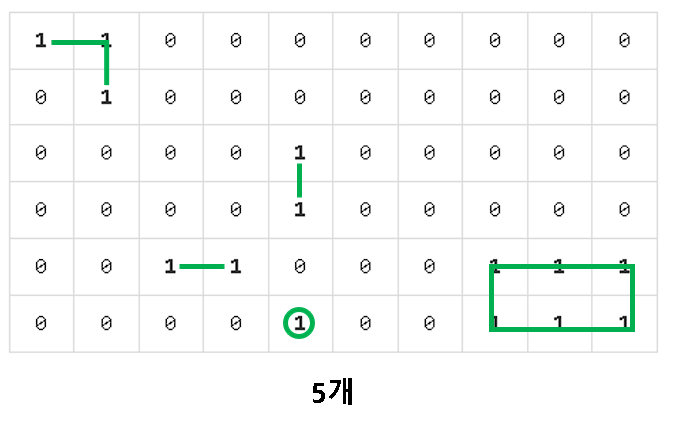
자동 생성된 설명 

It was difficult to find the rule. So, it should be analyzed with a lot of test cases.

The point is “**square root**”. It can calculate better and find solution easily.

|  |
| --- |
| **import** **sys**  **from** **math** **import** sqrt *# \*\* 0.5 계산으로 대체 가능*  inputF = sys.stdin.readline  dist = []  t = int(inputF())  **for** \_ **in** range(t):  num1, num2 = map(int, inputF().split())  dist.append(num2 – num1)  **for** dis **in** dist: *# 각각의 테스트 케이스 한 번에 처리*  y = int(sqrt(dis)) *# 거리의 제곱근*  z = y + 1  **if** y == 1: *# 규칙1 : 제곱근이 1이면 (거리만큼 출력)*  print(dis)  **elif** dis >= y \* z + 1: *# 규칙2 : 거리가 N\*(N+1)+1 이상일 때 (같은 제곱근 범위 안)*  print(y + z)  **elif** dis >= y \*\* 2 + 1: *# 규칙3 : 거리가 N^2+1 이상일 때 (새로운 제곱근 범위)*  print(y \* 2)  **else**:  print(y \* 2 – 1) |

Q 1012. 유기농 배추



Count if the field value is ‘1’. But there are ‘1’s around ‘1’, need to create a group.

Example dataset is ‘2D’ array. And, need to find solution using a sequential search.

= Can use **DFS** or **BFS** algorithm.

1. Check all values in the array. Using DFS.
2. DFS algorithm

* A. If the point has ‘1’, use a recursive DFS to check EWNS(East, West…)
* B. If the point doesn‘t exist(Out of range) or the point has ‘0’, return.

★ In python, **runtime error** may occur if not use ‘setrecursionlimit()’.

|  |
| --- |
| **import** **sys**  sys.setrecursionlimit(10\*\*6) *# 재귀 제한에 의한 런타임 오류 방지*  inputF = sys.stdin.readline *# 입출력 속도 개선*  **def** DFS(field, x, y):  *# 인덱스가 벗어나는 경우에 즉시 return*  **if** x < 0 **or** y < 0 **or** x >= N **or** y >= M:  **return** 0  *# 해당 위치가 1이라면 인접한 곳(상하좌우)에 1이 있는지 재귀(DFS)*  **elif** field[x][y] == 1:  field[x][y] = 0  DFS(field, x-1, y)  DFS(field, x+1, y)  DFS(field, x, y-1)  DFS(field, x, y+1)  **return** 1  T = int(inputF())  **for** \_ **in** range(T):  *# M:가로(열), N:세로(행), K:배추개수*  M, N, K = map(int, inputF().split())  *# 배추밭*  field = [[0 **for** col **in** range(M)] **for** \_ **in** range(N)]  *# 배추 위치*  **for** i **in** range(K):  *# x:열, y:행*  x, y = map(int, inputF().split())  field[y][x] = 1  total = 0  **for** i **in** range(N): *# 세로 반복*  **for** j **in** range(M): *# 가로 반복*  **if** DFS(field, i, j):  total += 1  print(total)  *# 그래프 이론, 깊이 우선 탐색(DFS), 너비 우선 탐색(BFS)* |

Q .

Hello