# 解決済み

A-E

# A Strictly Increasing?

正解

# B Make Target

正解

でも無駄が多いので(３重ループ)改善したコード(２重ループ)を書いた。改善したコードは端からの距離によって色が決まるので４つの距離を求め、それを利用する。

## 正解コード

N = int(input())

grid = [[None]\*N for \_ in range(N)]

for i in range(1,N//2+2):

j = N+1-i

if i % 2 == 1:

for y in range(i-1, j):

for x in range(i-1, j):

grid[y][x] = "#"

else:

for y in range(i-1, j):

for x in range(i-1, j):

grid[y][x] = "."

for i in range(N):

print(\*grid[i], sep = "")

## 改善コード

N = int(input())

grid = [[None]\*N for \_ in range(N)]

for i in range(N):

for j in range(N):

d = min(i+1, j+1, N-i, N-j)

if d%2 == 1:

grid[i][j] = "#"

else:

grid[i][j] = "."

for i in range(N):

print(\*grid[i], sep = "")

# C Shortest Duplicate Subarray

正解

解説と大体同じ

自分の方が速かった。

# D Pigeon Swap

正解

全く同じ考え方。だけど解説の”ラベル鳩”という呼び方がイマイチ。

# E Flip Edge

不正解

この問題は拡張ダイクストラ法。

(不正解コード)

自分のコードはWA１で他はACだった。このコードの間違いは、あるノードまでの最短経路を通常状態とリバース状態を分けずに一緒にしていること。そのため、あるノードsに辿り着く時、通常状態で入った場合のコストとリバース状態で入った時のコストを別にすること。

(不正解コード2)

これはコードは間違っていないがTLEを起こす方法である。ダイクストラとして不完全で確定したノードを記録していない。そのため、すでに確定しているノードに対してもう一度無駄な探索を入れることになる。

(正解コード)

## 不正解コード

from heapq import heapify, heappop, heappush

N, M, X = map(int, input().split())

ad = [[] for \_ in range(N)]

r\_ad = [[] for \_ in range(N)]

visited = [float("inf")]\*N

visited[0] = 0

for i in range(M):

u, v = map(int, input().split())

u -= 1

v -= 1

ad[u].append(v)

r\_ad[v].append(u)

q = [(0, 0, 0)]

heapify(q)

while q:

cost, node, r\_mode = heappop(q)

if r\_mode == 0:

nc = cost+1

for nxt in ad[node]:

if visited[nxt] < nc:

continue

heappush(q, (nc, nxt, 0))

visited[nxt] = nc

nc = cost+X+1

for nxt in r\_ad[node]:

if visited[nxt] < nc:

continue

heappush(q, (nc, nxt, 1))

visited[nxt] = nc

else:

nc = cost+1

for nxt in r\_ad[node]:

if visited[nxt] < nc:

continue

heappush(q, (nc, nxt, 1))

visited[nxt] = nc

nc = cost+X+1

for nxt in ad[node]:

if visited[nxt] < nc:

continue

heappush(q, (nc, nxt, 0))

visited[nxt] = nc

print(visited[N-1])

## 不正解コードTLE

from heapq import heapify, heappop, heappush

N, M, X = map(int, input().split())

ad = [[] for \_ in range(N)]

r\_ad = [[] for \_ in range(N)]

visited = [float("inf")]\*N

r\_visited = [float("inf")]\*N

visited[0] = 0

r\_visited[0] = X

for i in range(M):

u, v = map(int, input().split())

u -= 1

v -= 1

ad[u].append(v)

r\_ad[v].append(u)

q = [(0, 0, 0)]

heapify(q)

while q:

cost, node, r\_mode = heappop(q)

if r\_mode == 0:

r\_visited[node] = min(r\_visited[node], cost+X)

nc = cost+1

for nxt in ad[node]:

if visited[nxt] < nc:

continue

visited[nxt] = nc

heappush(q, (nc, nxt, 0))

nc = cost+X+1

for nxt in r\_ad[node]:

if r\_visited[nxt] < nc:

continue

r\_visited[nxt] = nc

heappush(q, (nc, nxt, 1))

else:

visited[node] = min(visited[node], cost+X)

nc = cost+1

for nxt in r\_ad[node]:

if r\_visited[nxt] < nc:

continue

r\_visited[nxt] = nc

heappush(q, (nc, nxt, 1))

nc = cost+X+1

for nxt in ad[node]:

if visited[nxt] < nc:

continue

visited[nxt] = nc

heappush(q, (nc, nxt, 0))

print(min(visited[N-1], r\_visited[N-1]))

## 正解コード

from heapq import heapify, heappop, heappush

N, M, X = map(int, input().split())

ad = [[] for \_ in range(N)]

r\_ad = [[] for \_ in range(N)]

for i in range(M):

u, v = map(int, input().split())

u -= 1

v -= 1

ad[u].append(v)

r\_ad[v].append(u)

v = [float("inf")]\*N

r\_v = [float("inf")]\*N

con = [False]\*N

r\_con = [False]\*N

goal = N-1

q = [(0, 0, 0)]

heapify(q)

while q:

cost, node, r\_mode = heappop(q)

if r\_mode == 0:

if node == goal and con[goal] == False:

con[goal] = True

v[goal] = cost

if r\_con[goal] != False:

break

else:

continue

if con[node]:

continue

con[node] = True

v[node] = cost

nc = cost+1

for nxt in ad[node]:

if not con[nxt] and v[nxt] > nc:

v[nxt] = nc

heappush(q, (nc, nxt, 0))

nc += X

for nxt in r\_ad[node]:

if not r\_con[nxt] and r\_v[nxt] > nc:

r\_v[nxt] = nc

heappush(q, (nc, nxt, 1))

else:

if node == goal and r\_con[goal] == False:

r\_con[goal] = True

r\_v[goal] = cost

if con[goal] != False:

break

else:

continue

if r\_con[node]:

continue

r\_con[node] = True

r\_v[node] = cost

nc = cost+1

for nxt in r\_ad[node]:

if not r\_con[nxt] and r\_v[nxt] > nc:

r\_v[nxt] = nc

heappush(q, (nc, nxt, 1))

nc += X

for nxt in ad[node]:

if not con[nxt] and v[nxt] > nc:

v[nxt] = nc

heappush(q, (nc, nxt, 0))

print(min(v[goal], r\_v[goal]))

# F Smooth Occlusion

解いてない

# G Minimum Steiner Tree 2

解いてない