# 復習済み

A-F

# A 22222

正解

# B cat

正解

# C Debug

正解

この問題で最も大事なのは文字列を切って合成する操作をしないということだった。これだけで800msから100msになる。

自分は前から文字を見ていって置換したが、そうすると一個前に戻って置換操作が終わってから再び文字列の後ろを見ていくので、置換すると同じ場所を2回見る無駄がある。

そこで改善コードとして後ろから置換するコードを書いた。

## 最初の正解コード (843 ms)

s = input()

n = len(s)

i = 0

count = 0

while i < n:

if s[i] == "W":

count += 1

i += 1

elif count > 0 and s[i] == "A":

s = s[:i-count] + "A" + "C"\*count + s[i+1:]

count = 0

i += 1

elif count > 0 and s[i] != "A":

count = 0

i += 1

else:

i += 1

print(s)

## 改善コード (101 ms )

s = list(input())

n = len(s)

for i in range(n-1, 0, -1):

if s[i-1] == "W" and s[i] == "A":

s[i-1] = "A"

s[i] = "C"

print(\*s, sep = "")

## 改善ミスコード(TLE)

s = input()

n = len(s)

for i in range(n-1, 0, -1):

if s[i-1:i+1] == "WA":

s = s[:i-1] + "AC" + s[i+1:]

print(s)

# D Colorful Bracket Sequence

正解

アルゴリズムも一緒

# E Palindromic Shortest Path

正解　(ただし、chatGPTを使った)

## 正解コード

from collections import deque

def shortest\_palindromic\_path(N, C):

INF = float('inf')

dist = [[INF] \* N for \_ in range(N)]

queue = deque()

for i in range(N):

dist[i][i] = 0

queue.append((i, i, 0))

for i in range(N):

for j in range(N):

if C[i][j] != "-" and i != j:

dist[i][j] = 1

queue.append((i, j, 1))

while queue:

u, v, d = queue.popleft()

if dist[u][v] < d:

continue

for nu in range(N):

if C[nu][u] != "-":

for nv in range(N):

if C[v][nv] == C[nu][u] and dist[nu][nv] > d + 2:

dist[nu][nv] = d + 2

queue.append((nu, nv, d+2))

for i in range(N):

print(" ".join(str(dist[i][j]) if dist[i][j] != INF else "-1" for j in range(N)))

N = int(input().strip())

C = [input().strip() for \_ in range(N)]

shortest\_palindromic\_path(N, C)

# F Alkane

解いてない

最初のコードはDFSによるものだが、これだと再帰に時間がかかる。他の人の回答を見た時、pypy限定のモジュールがあって

import pypyjit

pypyjit.set\_param('max\_unroll\_recursion=-1')

によって高速化されている。ただ不本意である。

よって別解として木dpのdp配列を用いたものを書いた。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 別解コード | 正解コード(DFS) | 別解の方が早い |
| hand00.txt | 70 | 57 |  |
| hand01.txt | 69 | 56 |  |
| hand02.txt | 70 | 57 |  |
| hand03.txt | 71 | 57 |  |
| sample00.txt | 70 | 58 |  |
| sample01.txt | 70 | 57 |  |
| sample02.txt | 70 | 57 |  |
| testcase00.txt | 71 | 58 |  |
| testcase01.txt | 69 | 58 |  |
| testcase02.txt | 68 | 58 |  |
| testcase03.txt | 71 | 58 |  |
| testcase04.txt | 342 | 334 |  |
| testcase05.txt | 347 | 354 | True |
| testcase06.txt | 184 | 183 |  |
| testcase07.txt | 348 | 380 | True |
| testcase08.txt | 175 | 197 | True |
| testcase09.txt | 346 | 407 | True |
| testcase10.txt | 210 | 258 | True |
| testcase11.txt | 365 | 609 | True |
| testcase12.txt | 216 | 215 |  |
| testcase13.txt | 380 | 372 |  |
| testcase14.txt | 297 | 289 |  |
| testcase15.txt | 363 | 374 | True |
| testcase16.txt | 325 | 324 |  |
| testcase17.txt | 366 | 390 | True |
| testcase18.txt | 357 | 371 | True |
| testcase19.txt | 350 | 356 | True |
| testcase20.txt | 234 | 219 |  |
| testcase21.txt | 377 | 361 |  |
| testcase22.txt | 240 | 357 | True |
| testcase23.txt | 325 | 522 | True |
| testcase24.txt | 300 | 321 | True |
| testcase25.txt | 301 | 343 | True |
| testcase26.txt | 223 | 212 |  |
| testcase27.txt | 333 | 338 | True |
| testcase28.txt | 294 | 303 | True |
| testcase29.txt | 324 | 379 | True |
| testcase30.txt | 269 | 340 | True |
| testcase31.txt | 282 | 307 | True |
| testcase32.txt | 226 | 209 |  |
| testcase33.txt | 384 | 382 |  |
| testcase34.txt | 156 | 150 |  |
| testcase35.txt | 385 | 366 |  |
| testcase36.txt | 199 | 228 | True |
| testcase37.txt | 348 | 373 | True |
| testcase38.txt | 233 | 258 | True |
| testcase39.txt | 364 | 377 | True |
| testcase40.txt | 310 | 309 |  |
| testcase41.txt | 339 | 333 |  |
| testcase42.txt | 195 | 198 | True |
| testcase43.txt | 359 | 350 |  |
| testcase44.txt | 267 | 290 | True |
| testcase45.txt | 381 | 356 |  |
| testcase46.txt | 180 | 197 | True |
| testcase47.txt | 397 | 361 |  |
| testcase48.txt | 176 | 169 |  |
| testcase49.txt | 416 | 357 |  |
| testcase50.txt | 361 | 354 |  |
| testcase51.txt | 428 | 352 |  |

## 正解コード (DFS, pypyによる再帰高速化)

import sys

sys.setrecursionlimit(10\*\*8)

import pypyjit

pypyjit.set\_param('max\_unroll\_recursion=-1')

n = int(input())

adjacent = [[] for \_ in range(n)]

for \_ in range(n-1):

a, b = map(int, input().split())

a -= 1

b -= 1

adjacent[a].append(b)

adjacent[b].append(a)

def dfs(child, parent):

tmp = []

global ans

for node in adjacent[child]:

if node == parent:

continue

tmp.append(dfs(node, child))

if parent == -1:

if len(tmp) >= 4:

tmp.sort(reverse = True)

ans = max(ans, sum(tmp[:4]))

else:

if len(tmp) < 3:

return 1

else:

tmp.sort(reverse = True)

tmp\_ans = sum(tmp[:3]) + 1

ans = max(ans, tmp\_ans)

if len(tmp) >= 4:

ans = max(ans, sum(tmp[:4]))

return tmp\_ans

ans = 0

dfs(0, -1)

if ans >= 4:

print(ans+1)

else:

print(-1)

## 別解コード (木DP)

from collections import deque

n = int(input())

adjacent = [[] for \_ in range(n)]

for \_ in range(n-1):

a, b = map(int, input().split())

a -= 1

b -= 1

adjacent[a].append(b)

adjacent[b].append(a)

dp = [None]\*n

stack = deque([(0, -1)])

ans = 0

while stack:

parent, root = stack.pop()

tmp = []

flag = False

for child in adjacent[parent]:

if child == root:

continue

if dp[child] == None:

stack.append((parent, root))

stack.append((child, parent))

flag = True

break

else:

tmp.append(dp[child])

if flag:

continue

if parent == 0:

if len(tmp) >= 4:

tmp.sort(reverse = True)

dp[parent] = sum(tmp[:4])

else:

dp[parent] = 1

else:

if len(tmp) < 3:

dp[parent] = 1

else:

tmp.sort(reverse = True)

dp[parent] = sum(tmp[:3])+1

if len(tmp) >= 4:

ans = max(ans, sum(tmp[:4]))

ans = max(ans, max(dp))

if ans >= 4:

print(ans+1)

else:

print(-1)

# G Dense Buildings

解いてない。

正解コードは解説のコードをpythonにしたもの。どの高さまで合成(union)したら始点と終点がつながるかをqueryごとに並行二分探索する。

大きなポイントはpairsとmidでビルの高さごとに分ける時defaultdictは使わず２次元配列を使うこと。nが大きくなるとdefaultdictのようなバラバラのメモリ配置になるものより、連続した２次元配列の方がキャッシュミスが起こりづらいため、早い。

## 正解コード

from collections import defaultdict

class Unionfind:

def \_\_init\_\_(self, n):

self.A = [-1]\*n

def find(self, x):

if self.A[x] < 0:

return x

else:

self.A[x] = self.find(self.A[x])

return self.A[x]

def union(self, a, b):

ra = self.find(a)

rb = self.find(b)

if ra == rb:

return False

if self.A[ra] > self.A[rb]:

self.A[ra] = rb

elif self.A[ra] < self.A[rb]:

self.A[rb] = ra

else:

self.A[ra] -= 1

self.A[rb] = ra

return True

H, W = map(int, input().split())

F = []

for \_ in range(H):

F.append(list(map(int, input().split())))

Q = int(input())

queries = [None]\*Q

for q in range(Q):

a, b, y, c, d, z = map(int, input().split())

a -= 1

b -= 1

c -= 1

d -= 1

queries[q] = (a\*W+b, y, c\*W+d, z)

pairs = [[] for \_ in range(10\*\*6+1)]

for i in range(H-1):

for j in range(W):

min\_f = min(F[i][j], F[i+1][j])

pairs[min\_f].append((i\*W+j, (i+1)\*W+j))

for i in range(H):

for j in range(W-1):

min\_f = min(F[i][j], F[i][j+1])

pairs[min\_f].append((i\*W+j, i\*W+j+1))

L = [1]\*Q

R = [10\*\*6+1]\*Q

HW = H\*W

while True:

mid = [[] for \_ in range(10\*\*6+1)]

flag = True

for q in range(Q):

if R[q]-L[q] <= 1:

continue

flag = False

mid[(L[q]+R[q])//2].append(q)

if flag:

break

uf = Unionfind(HW)

for i in range(10\*\*6, 0, -1):

for u, v in pairs[i]:

uf.union(u, v)

for q in mid[i]:

s, \_, t, \_ = queries[q]

if uf.find(s) == uf.find(t):

L[q] = i

else:

R[q] = i

for q in range(Q):

\_, y, \_, z = queries[q]

print(y+z-2\*min(L[q], y, z))