파이썬 코딩테스트 교재 3권 (기업코테 TOP)

기업 코테 빈출 TOP 패턴 + 대표 문제 30선(해설/코드 포함)

포지션/회사에 따라 편차는 있지만, 아래 패턴은 거의 모든 대형 코테에서 반복됩니다.

목차

1. 코테 전략

- 문제 1개당 ① 설계 ② 구현 ③ 검증 3단계로 분리한다.
- 입력이 크면 O(n^2) 의심. 제한을 보고 복잡도부터 가늠한다.
- 엣지케이스를 먼저 적는다: 빈 배열, 길이 1, 전부 동일, 최댓값, 음수, 오버플로(파이썬은 안전).
- 제출 전: 샘플 외에 최소 3개 자가 테스트를 만든다.

2. 구현(시뮬레이션) TOP 체크리스트

- 좌표계/방향: (dr,dc) 배열을 만들어 인덱스로 관리
- 경계 검사: 0<=r<n, 0<=c<m 형태로 함수화
- 상태를 '불변'으로 다루기: 다음 상태(next)를 만들어 갱신
- 정렬 기준이 복잡하면 key 튜플로 명확히

```
# 4방향 이동 템플릿
dr = (-1, 1, 0, 0)
dc = (0, 0, -1, 1)

def inside(r, c, n, m):
    return 0 <= r < n and 0 <= c < m
```

3. 문자열/파싱/정규화

기업 코테는 '규칙에 맞게 문자열을 파싱하고, 정규화한 뒤, 조건대로 출력'이 자주 나옵니다.

- 대소문자 통일: s.lower()
- 공백/특수문자 처리: strip, replace, split
- 토큰화: split 후 for로 규칙 적용
- 숫자/문자 혼합: 한 글자씩 훑으며 상태 머신처럼 처리

```
# 예: 숫자 토큰만 뽑기(연속 숫자)
s = input().rstrip()
nums = []
cur = ""
for ch in s:
    if ch.isdigit():
        cur += ch
    else:
        if cur:
            nums.append(int(cur))
            cur = ""
if cur:
    nums.append(int(cur))
print(nums)
```

4. 인터벌/스위핑/라인스위프

구간 겹침/최대 동시 진행/병합은 대형 코테에서 매우 빈출입니다.

```
# 인터벌 병합
intervals.sort()
merged = []
for s, e in intervals:
    if not merged or merged[-1][1] < s:
        merged.append([s, e])
    else:
        merged[-1][1] = max(merged[-1][1], e)

# 라인스위프: 최대 동시 개수
events = []
for s, e in intervals:
    events.append((s, +1))
    events.append((e, -1))   # [s,e) 반개구간 가정
events.sort()
cur = best = 0
for _, d in events:
    cur += d
    best = max(best, cur)
print(best)
```

5. 그래프 TOP

- 다익스트라: (가중치>=0) 최단거리 기본
- 0-1 BFS: 가중치가 0 또는 1일 때
- 위상정렬: 선후관계(의존성) + DAG
- 유니온파인드: 연결/사이클/최소 스패닝 트리

```
# 위상정렬(Kahn)
from collections import deque
q = deque([i for i in range(n) if indeg[i] == 0])
order = []
while q:
    v = q.popleft()
    order.append(v)
    for nv in g[v]:
        indeg[nv] -= 1
        if indeg[nv] == 0:
            q.append(nv)
```

6. 트리 TOP

트리는 DFS 한 번으로 '부모/깊이/서브트리 크기'를 뽑는 것이 기본입니다.

```
sys.setrecursionlimit(300000)

parent = [-1]*n
depth = [0]*n

def dfs(root):
    stack = [(root, -1)]
    parent[root] = -1
    depth[root] = 0
    while stack:
        v, p = stack.pop()
        for nv in tree[v]:
            if nv == p:
                continue
            parent[nv] = v
            depth[nv] = depth[v] + 1
            stack.append((nv, v))
```

LCA(최소 공통 조상)는 이진 점프(booster table) 형태가 빈출입니다.

7. 펜윅 트리(빈출 형태만)

구간합/점 업데이트가 많이 나오면 펜윅(BIT)이 간단하고 빠릅니다.

```
class Fenwick:
    def __init__(self, n):
        self.n = n
        self.bit = [0]*(n+1)

    def add(self, i, delta):  # 1-indexed
        while i <= self.n:
            self.bit[i] += delta
            i += i & -i

    def sum(self, i):
        s = 0
        while i > 0:
            s += self.bit[i]
            i -= i & -i
        return s

    def range_sum(self, l, r):
        return self.sum(r) - self.sum(l-1)
```

8. TOP 30 문제 리스트

〈b〉T01〈/b〉 시뮬레이션: 로봇이 명령 문자열대로 이동, 벽/장애물 처리

〈b〉T02〈/b〉 문자열 파싱: 로그 분석(시간/레벨/메시지) 후 조건 필터

〈b〉T03〈/b〉 스위핑: 최대 동시 회의 수

〈b〉T04〈/b〉 인터벌 병합 후 총 길이

〈b〉T05〈/b〉 투포인터: 조건 만족하는 최소 구간

〈b〉T06〈/b〉 해시: 두 배열의 교집합/차집합 카운팅

〈b〉T07〈/b〉 그리디: 최소 비용으로 작업 스케줄링

〈b〉T08〈/b〉 이분탐색: 최소 가능한 최대값(파라메트릭)

〈b〉T09〈/b〉 BFS: 격자 최단거리(벽 1개까지 부수기)

〈b〉T10〈/b〉 0-1 BFS: 비용 0/1 격자

〈b〉T11〈/b〉 다익스트라: 노드 수 2e5 근처 최단경로

〈b〉T12〈/b〉 최단경로: 여러 출발점에서 한 번에

〈b〉T13〈/b〉 위상정렬: 선수과목 + 가능 여부/순서 출력

〈b〉T14〈/b〉 유니온파인드: 사이클 판별

〈b〉T15〈/b〉 MST: 최소 스패닝 트리(크루스칼)

〈b〉T16〈/b〉 DP: 문자열 편집(단순화 버전)

〈b〉T17〈/b〉 DP: 분할/최적 괄호(기초)

〈b〉T18〈/b〉 LIS 변형: 최대 증가/감소 수열

〈b〉T19〈/b〉 트리: 서브트리 크기 질의

〈b〉T20〈/b〉 트리: 두 노드 거리(부모/깊이)

〈b〉T21〈/b〉 LCA: 다중 질의

〈b〉T22〈/b〉 펜윅: 점 업데이트 + 구간합 Q

〈b〉T23〈/b〉 세그트리: 구간 최솟값 Q(기초)

〈b〉T24〈/b〉 스택: 히스토그램 최대 직사각형

〈b〉T25〈/b〉 문자열: KMP로 패턴 찾기(맛보기)

〈b〉T26〈/b〉 그래프: SCC(코사라주) 맛보기

〈b〉T27〈/b〉 이분 매칭(기초) 맛보기

〈b〉T28〈/b〉 비트마스킹: 부분집합 DP(작은 n)

〈b〉T29〈/b〉 좌표 압축 + 펜윅 응용

〈b〉T30〈/b〉 종합: 조건이 많은 구현 + 자료구조 결합

9. TOP 30 해설/코드(대표 12문제)

기업 코테 대비용으로 대표 12문제는 비교적 '완성형 코드'를 제공합니다.

T03 최대 동시 회의 수 (라인스위프)

```
n = int(input())
events = []
for _ in range(n):
    s, e = map(int, input().split())
    events.append((s, 1))
    events.append((e, -1))   # [s,e)
events.sort()

cur = best = 0
for _, d in events:
    cur += d
    if cur > best:
        best = cur
print(best)
```

T09 벽 1개까지 부수기 BFS

```
from collections import deque
n, m = map(int, input().split())
a = [list(map(int, list(input().rstrip()))) for _ in range(n)]
dist = [[[-1]*2 for _ in range(m)] for _ in range(n)]
q = deque([(0,0,0)])
dist[0][0][0] = 1

dr = (-1,1,0,0); dc=(0,0,-1,1)
while q:
    r,c,b = q.popleft()
    for k in range(4):
        nr, nc = r+dr[k], c+dc[k]
        if not (0<=nr<n and 0<=nc<m):
            continue
        nb = b
        if a[nr][nc] == 1:
            if b == 1:
                continue
            nb = 1
        if dist[nr][nc][nb] == -1:
            dist[nr][nc][nb] = dist[r][c][b] + 1
            q.append((nr,nc,nb))

ans0 = dist[n-1][m-1][0]
ans1 = dist[n-1][m-1][1]
cands = [x for x in (ans0, ans1) if x != -1]
print(min(cands) if cands else -1)
```

T11 다익스트라(대형 입력)

```
import heapq
n, m = map(int, input().split())
g = [[] for _ in range(n)]
for _ in range(m):
    a, b, w = map(int, input().split())
```

```
    a -= 1; b -= 1
    g[a].append((b, w))
    g[b].append((a, w))

s, t = map(int, input().split())
s -= 1; t -= 1

INF = 10**18
dist = [INF]*n
dist[s] = 0
pq = [(0, s)]
while pq:
    d, v = heapq.heappop(pq)
    if d != dist[v]:
        continue
    if v == t:
        break
    for nv, w in g[v]:
        nd = d + w
        if nd < dist[nv]:
            dist[nv] = nd
            heapq.heappush(pq, (nd, nv))
print(dist[t] if dist[t] < INF else -1)
```

T13 위상정렬(순서 출력)

```
from collections import deque
n, m = map(int, input().split())
g = [[] for _ in range(n)]
ind = [0]*n
for _ in range(m):
    a, b = map(int, input().split())
    a -= 1; b -= 1
    g[a].append(b)
    ind[b] += 1

q = deque([i for i in range(n) if ind[i] == 0])
order = []
while q:
    v = q.popleft()
    order.append(v)
    for nv in g[v]:
        ind[nv] -= 1
        if ind[nv] == 0:
            q.append(nv)

if len(order) != n:
    print("IMPOSSIBLE")
else:
    print(*[x+1 for x in order])
```

T15 MST(크루스칼)

```
n, m = map(int, input().split())
edges = []
for _ in range(m):
    a, b, w = map(int, input().split())
    edges.append((w, a-1, b-1))
edges.sort()
```

```
parent = list(range(n))
rank = [0]*n

def find(x):
    while parent[x] != x:
        parent[x] = parent[parent[x]]
        x = parent[x]
    return x

def union(a, b):
    ra, rb = find(a), find(b)
    if ra == rb:
        return False
    if rank[ra] < rank[rb]:
        ra, rb = rb, ra
    parent[rb] = ra
    if rank[ra] == rank[rb]:
        rank[ra] += 1
    return True

cost = 0
cnt = 0
for w, a, b in edges:
    if union(a, b):
        cost += w
        cnt += 1
        if cnt == n-1:
            break
print(cost if cnt == n-1 else "IMPOSSIBLE")
```

T22 펜윅: 점 업데이트/구간합

```
n, q = map(int, input().split())
arr = [0] + list(map(int, input().split()))

class Fenwick:
    def __init__(self, n):
        self.n = n
        self.bit = [0]*(n+1)
    def add(self, i, delta):
        while i <= self.n:
            self.bit[i] += delta
            i += i & -i
    def sum(self, i):
        s = 0
        while i > 0:
            s += self.bit[i]
            i -= i & -i
        return s
    def range_sum(self, l, r):
        return self.sum(r) - self.sum(l-1)

fw = Fenwick(n)
for i in range(1, n+1):
    fw.add(i, arr[i])

for _ in range(q):
    t, a, b = map(int, input().split())
    if t == 1:  # update index a to value b
```

```
        delta = b - arr[a]
        arr[a] = b
        fw.add(a, delta)
    else:       # query sum [a,b]
        print(fw.range_sum(a, b))
```

나머지 TOP 문제들은 2권 패턴을 조합한 형태입니다. 이 책의 코드를 '뼈대'로 두고, 문제 조건에 맞게 변형하는 훈련을 하세요.