파이썬 코딩테스트 교재 2권 (실전)

실전 알고리즘 패턴 12종 + 유형별 문제/풀이

목표: 문제를 읽고 '어떤 패턴으로 푸는지'를 바로 떠올리는 수준

목차

1. 코테 공통 템플릿

아래 템플릿을 기본으로 시작하면 실수를 크게 줄일 수 있습니다.

```
import sys
from collections import deque, defaultdict, Counter
import heapq
input = sys.stdin.readline

def solve():
    # TODO: 구현
    pass

if __name__ == "__main__":
    solve()
```

디버깅 팁: ① 작은 입력으로 손으로 검증 ② 중간 상태를 print로 확인 ③ 경계값(n=0,1,최대) 테스트

2. 누적합/차분 배열

구간합이 많으면 O(n)으로 미리 누적합을 만들어 질의마다 O(1)로 답합니다.

```
# 1-indexed prefix sum
ps = [0]*(n+1)
for i in range(1, n+1):
    ps[i] = ps[i-1] + arr[i-1]

# [l,r] 구간합 (1-indexed)
ans = ps[r] - ps[l-1]
```

차분 배열(diff)은 '구간에 +v' 업데이트가 많을 때 유용합니다.

```
diff = [0]*(n+2)
# [l,r]에 +v
diff[l] += v
diff[r+1] -= v
# 복원
cur = 0
for i in range(1, n+1):
    cur += diff[i]
    arr[i] = cur
```

3. 투 포인터/슬라이딩 윈도우

정렬되어 있거나, '연속 구간' 조건이 있을 때 자주 등장합니다.

```
# 합이 K 이상이 되는 최소 길이(양수 배열 가정)
l = 0
s = 0
best = 10**18
for r in range(n):
    s += arr[r]
    while s >= K:
        best = min(best, r-l+1)
        s -= arr[l]
        l += 1
```

4. 해시/카운팅/빈도

딕셔너리/Counter로 빈도를 만들고, 조건에 맞게 조회합니다.

```
from collections import Counter
cnt = Counter(arr)
most = max(cnt.items(), key=lambda x: (x[1], -x[0]))  # (값, 빈도) 예시
```

5. 정렬 + 그리디

그리디는 '정렬 후 한 번 훑기' 형태가 많습니다.

```
# 회의실 배정(끝나는 시간 기준 정렬)
meetings.sort(key=lambda x: (x[1], x[0]))
end = -1
ans = 0
for s, e in meetings:
    if s >= end:
        ans += 1
        end = e
```

6. 이분 탐색 (파라메트릭 서치)

정답을 '결정 문제(가능/불가능)'로 바꾸면 이분 탐색이 됩니다.

```
def ok(x):
    # x가 정답 후보일 때 조건을 만족하면 True
    return True

lo, hi = 0, 10**9
ans = hi
while lo <= hi:
    mid = (lo + hi)//2
    if ok(mid):
        ans = mid
        hi = mid - 1
    else:
        lo = mid + 1
print(ans)
```

7. 스택/큐/덱

스택: 괄호/오큰수/단조 스택. 덱: BFS, 슬라이딩 윈도우에 자주 사용.

```
from collections import deque
dq = deque()
dq.append(1)
dq.appendleft(0)
dq.pop()
dq.popleft()
```

8. 힙(우선순위 큐)

최솟값/최댓값을 반복적으로 뽑는 문제에서 사용합니다.

```
import heapq
hq = []
heapq.heappush(hq, 5)
heapq.heappush(hq, 2)
print(heapq.heappop(hq))  # 2
# 최대 힙은 -값으로
```

9. DFS/BFS 그래프 탐색

격자/그래프에서 연결 요소, 최단 거리(가중치 1)를 찾을 때 BFS.

```
from collections import deque

def bfs(start):
    dist = [-1]*n
    q = deque([start])
    dist[start] = 0
    while q:
        v = q.popleft()
        for nv in g[v]:
            if dist[nv] == -1:
                dist[nv] = dist[v] + 1
                q.append(nv)
    return dist
```

10. 최단경로 기초 (다익스트라)

가중치가 음수가 없으면 다익스트라가 표준입니다.

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

11. DP 핵심 패턴

DP는 '상태 정의'가 90%입니다. 보통 1차원/2차원으로 시작합니다.

```
# 1) 1차원 DP: dp[i] = i까지의 최적
dp = [0]*(n+1)
dp[0] = 0
for i in range(1, n+1):
    dp[i] = dp[i-1] + cost[i]

# 2) 배낭(0/1): dp[w]를 뒤에서 갱신
for weight, value in items:
    for w in range(W, weight-1, -1):
        dp[w] = max(dp[w], dp[w-weight] + value)
```

12. 유니온파인드

서로소 집합(DSU)은 '연결 여부'를 빠르게 관리합니다.

```
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
```

13. 실전 문제 60선

각 문제는 위 12개 패턴 중 하나 이상으로 풀리도록 구성했습니다.

누적합/차분

- S01 1차원 구간합 질의
- S02 2차원 구간합(누적합 표)
- S03 구간 업데이트 후 최종 배열

투포인터

- S04 합이 K 이상 최소 길이
- S05 서로 다른 원소 최대 길이
- S06 정렬된 배열에서 두 수의 합

해시/카운팅

- S07 가장 많이 나온 문자
- S08 아나그램 판별
- S09 부분수열 합 카운팅(해시)

정렬+그리디

- S10 회의실 배정
- S11 최소 비용 합치기
- S12 간격 스케줄링

이분탐색

- S13 공유기 설치
- S14 예산 상한
- S15 나무 자르기

스택/큐

- S16 올바른 괄호
- S17 오큰수
- S18 큐 시뮬레이션

힙

- S19 K번째 작은 수 스트림
- S20 작업 스케줄링
- S21 이중 우선순위큐

BFS/DFS

- S22 연결 요소 개수
- S23 미로 최단거리
- S24 섬의 개수(격자)

다익스트라

- S25 최단거리 1회
- S26 경로 복원
- S27 여러 시작점 최단거리

DP

- S28 계단 오르기
- S29 LIS(가장 긴 증가 부분수열)
- S30 동전 교환

나머지 S31~S60은 위 패턴을 섞은 '혼합형'으로, 풀이 아이디어를 14장에서 제공합니다.

14. 해설/참고 코드

대표 문제 12개만 풀이 코드(정답 템플릿)를 제공합니다. 나머지는 같은 패턴으로 확장하세요.

S02 2차원 누적합

```
# n x m, q queries: sum of rectangle (r1,c1)-(r2,c2) 1-indexed
n, m = map(int, input().split())
a = [list(map(int, input().split())) for _ in range(n)]
ps = [[0]*(m+1) for _ in range(n+1)]
for i in range(1, n+1):
    row_sum = 0
    for j in range(1, m+1):
        row_sum += a[i-1][j-1]
        ps[i][j] = ps[i-1][j] + row_sum

q = int(input())
for _ in range(q):
    r1,c1,r2,c2 = map(int, input().split())
    ans = ps[r2][c2] - ps[r1-1][c2] - ps[r2][c1-1] + ps[r1-1][c1-1]
    print(ans)
```

S06 정렬된 배열 두 수의 합

```
arr.sort()
l, r = 0, n-1
best = None
while l < r:
    s = arr[l] + arr[r]
    if s == K:
        best = (arr[l], arr[r])
        break
    if s < K:
        l += 1
    else:
        r -= 1
print(best if best else "NONE")
```

S17 오큰수(단조 스택)

```
n = int(input())
a = list(map(int, input().split()))
ans = [-1]*n
st = []
for i, x in enumerate(a):
    while st and a[st[-1]] < x:
        ans[st.pop()] = x
    st.append(i)
print(*ans)
```

S29 LIS (O(n log n))

```
import bisect
tails = []
for x in arr:
    i = bisect.bisect_left(tails, x)
    if i == len(tails):
        tails.append(x)
```

```
    else:
        tails[i] = x
print(len(tails))
```

3권에서는 기업 코테에서 특히 자주 나오는 '상위 패턴 + 고난도 구현'을 집중적으로 다룹니다.