
탐욕 알고리즘 문제 해결을 위한 4가지 Tip

1. 문제 유형 파악하기 - "최적화 문제인가?"

가장 먼저, 문제의 요구사항이 **최적화** 문제인지 확인해야 합니다.

- **키워드 확인**: 문제에서 **"최소", "최대", "최소 횟수", "최대한 많은"** 과 같은 단어를 찾는 것이 첫 번째 신호입니다. 이는 여러 가능한 답 중에서 가장 좋거나 나쁜 '최적의 해'를 하나만 찾으라는 의미입니다.
- **되돌릴 필요가 없는가?**: "한 번 내린 선택을 나중에 다시 바꾸거나 되돌릴 필요 없이, 앞으로 나아가기만 해도 답이 나올까?" 라고 자문자답해 보세요. 만약 그렇다고 느껴진다면 그리디 문제일 가능성이 높습니다. (만약 "A를 선택했다가, 아니다 싶으면 B로 바꿔서 다시 시도해야" 한다면, 이는 백트래킹 문제입니다.)

2. 탐욕적 선택 기준(Greedy Choice) 찾기 - "무엇을 기준으로 정렬할까?"

그리디 문제임이 의심된다면, "매 순간 어떤 기준으로 선택하는 것이 가장 이득일까?"를 찾아야 합니다. 이 '기준'은 대부분 **입력 데이터를 정렬**하는 것에서부터 시작됩니다.

- **정렬 기준 찾기**: 문제의 핵심 가치를 나타내는 기준으로 데이터를 정렬하는 것이 핵심입니다.
    - **종료 시간**이 가장 빠른 순서 (예: `화물 도크` - 활동 선택 문제)
    - **무게/가치**가 가장 큰 순서 (예: `컨테이너 운반` - 배낭 문제 변형)
    - **마감 시간**이 임박한 순서 (예: 작업 스케줄링 문제)
    - **길이**가 짧거나 긴 순서

> **"이 문제의 데이터들을 어떤 기준으로 줄 세워야, 맨 앞(또는 맨 뒤)부터 처리하는 게 항상 최선일까?"** 를 고민하는 것이 그리디 알고리즘 해결의 90% !

3. 선택 증명해보기 - "이 선택에 후회는 없는가? (반례 찾기)"

나의 '탐욕적 선택'이 항상 올바른지 간단하게 검증하는 습관이 매우 중요합니다.

- **반례 찾아보기**: "지금 당장 가장 좋아 보이는 이 선택이, 나중에 더 큰 이득을 얻을 수 있는 기회를 막아버리지는 않을까?"를

생각해 보세요. 아주 작은 예시(`N=3` 또는 `4`)를 직접 손으로 그려보며, 나의 탐욕적인 선택이 정답으로 이어지지 않는 경우(반례)가 있는지 확인합니다.

- **반례가 존재한다면?**: 그 문제는 그리디 알고리즘으로 풀 수 없습니다. `요리사` 문제처럼, 모든 경우의 수를 따져봐야 하는 완전 탐색(백트래킹, DP 등) 문제일 가능성이 높습니다.

4. 코드 구현하기 - "정렬하고, 순회하고, 선택하기"

탐욕적 선택 기준을 찾았다면, 코드 구현은 보통 간단한 패턴을 따릅니다.

1. 입력 데이터를 **핵심 기준에 따라 정렬**합니다.
2. `for` 또는 `while` 반복문으로 정렬된 데이터를 **순회**합니다.
3. 현재 상태를 기준으로, 지금 보고 있는 데이터를 **선택할지 말지 결정**합니다.
4. 선택했다면, **정답과 상태 변수를 갱신**하고 다음으로 넘어갑니다.

[요약] 그리디 문제 해결 생각의 흐름

단계	질문	핵심 활동	예시 문제
1. 유형 파악	최소/최대 값을 찾는가? 선택을 되돌릴 필요가 없는가?	최적화 문제임을 인지하고 그리디 의심	`컨테이너 운반`, `화물 도크`
2. 기준 찾기	무엇을 기준으로 정렬해야 매번 가장 이득일까?	**종료 시간, 무게, 가치 등** 핵심 기준으로 데이터 정렬	`화물 도크` (종료 시간), `컨테이너 운반` (무게)
3. 증명/검증	내 선택이 미래의 더 좋은 선택을 막지는 않는가?	간단한 **반례**를 손으로 생각해보거나 그려보기	`요리사` (반례 존재 -> 그리디 X)
4. 구현	정렬 후, 어떻게 순회하며 선택을 누적할까?	`for` 또는 `while` 반복문으로 정렬된 데이터 처리	`컨테이너 운반` (가장 큰 것끼리 비교)