**그리디**

* 당장 좋은 것만 선택한다.
* 매 순간 가장 좋아보이는 것을 선택하며, 현재 선택이 나중에 미칠 영향에 대해서 고려하지 않는다.
* 가장 큰 순서대로, 가장 작은 순서대로 라는 기준을 알게 모르게 제시해준다.
* 해법이 정당한지 검토해야한다.

대표문제) 거스름돈

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명 텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* 화폐의 종류만큼 반복을 수행해야하기때문에, 위 코드의 시간 복잡도는 O(K)이다.
* 해법의 정당성: 가지고 있는 동전 중에서 큰 단위가 항상 작은 단위의 배수이므로 작은 단위의 동전들을 종합해 다른 해가 나올 수 없다.

**구현**

* 머릿속에 있는 알고리즘을 소스코드로
* 완전탐색, 시뮬레이션
* 리스트를 여러 개 선언하고, 그중에서 크기가 1,000만 이상인 리스트가 있다면 메모리 용량 제한으로 문제를 풀 수 없게 되는 경우도 있다.

대표문제) 상하좌우

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**DFS/BFS**

* 탐색이란 많은 양의 데이터 중에서 원하는 데이터를 찾는 과정이다. 대표적으로 DFS, BFS가 있다.
* 스택 자료구조를 활용하는 알고리즘은 재귀함수를 통해 간편하게 구현되는데, 대표적으로 DFS가 있다.
* 인접 리스트를 이용하자
* 그래프 탐색 문제가 나오면 DBS/BFS 떠올리자
* 양방향, 단방향 케이스 분류
* 그래프로 주어지는 경우, 배열로 주어지는 경우

인접리스트)

graph = [ [] for \_ in range(3) ]

# 노드 0에 연결된 노드 정보 저장 (노드, 거리)

graph[0].append((1,7))

graph[1].append((0,7))

**정렬**

* 파이썬 정렬 라이브러리를 사용하면 시간 복잡도 O(NlogN)을 보장한다.
* 튜플을 활용하자

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

**이진탐색**

* 시간 안에 찾는 문제
* 종료 조건: while start<=end
* Bisect: 정렬된 배열에서 특정한 원소를 찾아야할 때

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**그 외**