그리디 알고리즘

그리디 알고리즘은 말 그대로 탐욕스럽게 매 순간 최선의 선택을 하는 알고리즘이다. 이렇게 정의를 들어보면 구현하기가 굉장히 간단하고 편할 것 같지만, 어떤 최선의 선택을 하느냐에 따라 최종으로 구한 답이 최적의 답일 수도 있고 아닐 수도 있다. 따라서 그 선택이 최적해로 이어진다는 것을 증명하면 그 선택을 통해 알고리즘을 구현할 수 있는 것이다. 이제까지 배운 것을 정리해보자.

분할정복 : 보통 Top-down approach로 N개의 크기의 문제를 분할해가면서 푸는 방식. (겹치는 문제가 있음)
DP : 보통 Bottom-up approach로 분할정복과 비슷하게 풀지만 겹치는 문제가 없고 Memoization 사용.
Brute Force : 모든 경우에 대해 계산하기 때문에 항상 답을 구할 수는 있지만 효율이 최악.
그리디 : 매 순간 최선의 선택을 하여 최적해로 이어지기 때문에 위의 3개 방법보다 빠르고 모든 경우 계산 X.

일상생활에서 볼 수 있는 대표적인 그리디 알고리즘으로는 돈 거슬러주기 문제가 있다. 편의점에서 3750원 짜리 물건을 사는데 10,000원을 냈다고 하자. 거스름돈은 6250원이고 알바의 입장에서 돈을 거슬러줄 때는 직관적으로 5,000원 1개, 1,000원 1개, 100원 2개, 10원 5개를 줄 것이다. 이 방법이 가장 적은 수로 거슬러줄 수 있는 방법이기 때문이다. 이렇게 거슬러 준 이유를 생각해 보면 **가장 큰 단위의 돈**부터 계산하여 거슬러주기 때문인데, 바로 이 방식이 그리디 알고리즘이다. 이제 그리디 분야에서 꽤 이름있는 Activity Selection 문제를 살펴보자.

Activity Selection Problem

간단하게 설명해서 "활동 선택 문제"라고 하는데 기준에 따라 다르겠지만 일단 배운 방식으로는 활동을 가장 많이 할 수있는 방향으로 시간대를 선택하는 방법이다. 예를 들어, 회사에서 회의가 열리는 시간대가 쭉 나열되어 있는 시간표가 있다고 하자. 회의를 최대한 많이 참석하고자 하는데, 한번에 한개의 회의만 참석할 수 있기 때문에 하나의 회의에 참석했다면 반드시 그 회의가 끝나야 다음 회의에 참석할 수 있다. 이 때, 최대한 많은 회의를 참석할 수 있도록 회의를 고르는 알고리즘을 구현하는 것이 바로 이 문제의 해결법이다. 최대한 많은 회의를 참석하기 위해선 여러가지 기준을 생각할 수 있다. 회의 시간이 가장 짧은 것, 회의가 가장 빨리 시작하는 것, 회의가 가장 빨리 끝나는 것 등이 있겠지만 **회의가 가장 빨리 끝나는 것**이 회의에 참석할 수 있는 시간을 가장 많이 남겨주기 때문에 기준이 된다. 따라서 회의 시간표에 대해서 시작시간과 끝시간을 가지고 있는 회의를 끝시간이 작은 순으로 오름차순 정렬 시켜주고 compatible한 회의들을 전부 구하면 그게 답이 된다.

이제 조금 더 논리적으로 접근해 보자. S를 n개의 회의가 들어있는 집합이라고 하고 $S_{i,j}$를 a_i가 끝나고 시작해서 a_j가 시작하기 전에 끝나는 S의 부분집합이라고 하자. 우리는 a_0와 a_{n+1}이라는 가상 회의를 추가할 수 있는데 a_0는 f_0, 즉 시작하기 전에 끝나고, a_{n+1}은 $s_{n+1} = +\infty$, 즉 ∞의 시간이 지나야 시작한다. 즉 아무의미가 없는 회의를 집어넣음으로서 $S_{0,n+1}$을 모든 a_0가 끝나고 시작해서 a_{n+1}이 시작하기 전에 끝나는 S의 부분집합으로 정의할 수 있기 때문에 S에 들어있는 모든 회의 개수인 n이라고 할 수 있다.

다음으로 a_k를 $S_{i,j}$안에 있는 optimal한 회의라고 하면, $i < k < j$ 에 대해서 optimal schedule은 maximal subset of $S_{i,k}$, $\{a_k\}$, maximal subset of $S_{k,j}$라고 할 수 있다. 즉, a_k가 optimal하므로 a_k와 compatible 하지 않는(겹치는) 회의를 전부 제거할 수 있고 그 양쪽에 있는 $S_{i,k}$와 $S_{k,j}$에 똑같은 알고리즘을 적용해서 결국 참석할 수 있는 최대 회의 수를 구할 수 있다. c_{ij}를 $S_{i,j}$의 optimal schedule size라고 한다면, 다음과 같이 표현 가능하다.

$$c_{ij} = \begin{cases} 0 & \text{if } (S_{i,j} = \emptyset) \\ max_{i<j<k}(c_{ik} + c_{kj} + 1)(a_k \in S_{i,j}) & \text{if } (S_{i,j} \neq \emptyset) \end{cases}$$

위에서 설명한 대로 이해하면 간단한데, $S_{i,j}$에 어떤 회의도 없는 공집합의 경우 당연히 참석할 수 있는 회의는 없으므로 0이다. 다음은 a_k가 $S_{i,j}$안에 속했다면 optimal schedule이므로 일단 1개의 회의에 참석할 수 있는 것이다. 이제 나머지 양쪽의 schedule에서 optimal schedule을 다시 보면 되기 때문에 $c_{ik} + c_{kj}$를 해주는 것이다. 단, c_{ik}, c_{kj}를 구하는 방법에는 여러가지가 있을 수 있기 때문에 그 중 가장 큰 값을 구해주면 결과적으로 회의를 가장 많이 참석할 수 있는 수를 구할 수 있게 된다. 하지만 아까도 말했다시피 이렇게 선택하는 것이 최적해로 이어진다는 것을 증명해야 한다. 먼저 어떤 것을 증명해야 할지 theorem을 보자.

- $S_{i,j} \neq \emptyset$라고 하고, a_m을 $S_{i,j}$에서 가장 빨리 끝나는 회의라고 하자.
- a_m은 compatible한 회의의 집합 중 maximum-size를 가지는 여러 개의 집합에 반드시 속한다. 왜? a_m이 제일 빨리 끝나는 회의이기 때문에 자명하다.
- $S_{i,m} = \emptyset$이라고 하면, a_m을 선택하는 것으로 $S_{m,j}$만 남게 된다. 왜? a_m이 가장 일찍 끝나는데, a_m이 시작하기도 전에 끝나는 $S_{i,m}$이 존재하는 것은 말이 안된다.

이 Theorem이 증명될 경우, a_m은 반드시 $S_{i,j}$의 compatible한 maximum-size 집합에 들어가고 a_m을 선택하면 $S_{m,j}$가 남을 것이고, 거기서 다시 같은 방법으로 가장 빨리 끝나는 회의를 a'_m이라고 하면 a'_m도 집합에 들어가게 되어서 계속 가장 빨리 끝나는 회의를 기준으로 참석하기 때문에 최대한 많은 회의에 참석할 수 있게 된다.