완전 탐색 알고리즘이란 모든 경우의 수를 다 체크해서 정답을 찾는 방법
-> 무식하게 한다해서 brute force 라고도 부름

두 가지 규칙을 기본적으로 적용
1. 사용된 알고리즘이 적절한가? (문제를 해결할 수 있는가)
2. 효율적으로 동작하는가

또한 완전 탐색을 문제에 적용하기에 앞서 몇 가지를 판단해 보아야 합니다.
먼저 입력으로 주어지는 데이터의 크기를 보고 판단합니다. 완전탐색의 경우 n! 또는 2^N N^N의 경우가 많기 때문에 주어지는 크기가 아주 작아야합니다. 두 번째로 답의 범위가 매우 작기 때문에 임의의 답을 하나 선택하면 문제의 조건을 만족하는 역으로 추적이 가능합니다. 마지막으로 문제의 조건 중 하나를 고정하고 문제를 풀면 좀 더 간단하게 풀 수 있습니다.
다음으로 완전 탐색 기법들에 대해 소개를 하자면, 먼저 브루트 포스는 말 그대로 반복/조건문을 사용해서 모든 가능성을 확인하는 방법입니다. 가장 간단하지만 시간적으로는 효율적이지 못할 수 있습니다. 다음으로는 순열입니다. 순열은 n개의 원소 중 r개를 중복 없이 나열하는 것을 말합니다. 다음은 재귀함수로 함수 내에서 자기 자신을 호출하는 것을 말합니다. 다음은 비트 마스킹 방법으로 2진수 표현 기법을 활용합니다. 백트랙킹은 위에 설명된 기법들에 적용이 가능한 방법으로 탐색 중 현재 상태에서 가능한 후보군만 가지치기하며 탐색하는 방식입니다. 마지막으로 dfs/bfs가 있습니다.
순열에 대해 조금 자세히 설명을 드리면 시간 복잡도가 O(N!)이라서 10!이 360만이고 11!이 거의 4000만 정도라서 그 이상으로는 활용이 힘들겠네요. 보통 순열은 재귀나 반복문으로 구현 하는데 C++에는 algorithm 라이브러리에 next_permutation prev_permutation 함수가 있어서 조금 편하게 구할 수 있네요.
비트 마스킹 기법은 2진수를 이용하는 컴퓨터 연산을 이용하는 방법으로 예를 들면 5개의 원소로 된 배열을 2진수로 변경하면 2^5-1으로 31로 모든 경우의 수를 파악 할 수 있습니다.

그리디 알고리즘은 탐욕적 알고리즘이라고도 합니다. 탐욕적이라는 뜻 그대로 매 선택의 순간마다 당장 최적의 상황만을 선택합니다. 이는 최적해를 구하는 근사적인 방법이며 지역적으로는 최적의 해를 구했을 수 있어도 전역적으로 최적이라는 보장이 없습니다. 하지만 저희가 푸는 그리디 문제는 다 최적해를 찾을 수 있는 문제들이기 때문에 이 부분은 알고리즘 공부를 할 때는 걱정하지 않아도 될 것 같습니다. 그리디 문제 해결 방법/ 절차는 이런 순서를 따르게 됩니다. ~~
그리고 그리디로 문제를 해결하기 위해서는 다음 조건들을 만족해야합니다.
1. 즉 과거의 선택이 미래에 영향을 주면 안된다는 뜻입니다.
2. 부분 문제로 나누었을 때 각 부분 문제의 해결 방법이 최종 문제의 해결 방법이 된다는 뜻입니다.

그리디나 완탐의 경우는 사실 내용을 설명한다고 이해가 되는 부분들도 아니고 딱히 어떤 알고리즘을 활용해서 문제를 푸는게 아니여서,, 문제를 많이 풀어보는게 가장 효율적일 것 같습니다. 생각보다 정리할 내용이 많지는 않네요,,