### PA 2:

각각의 다양한 양이 담긴 포도주 잔 n개를 조건을 만족하며 가장 많이 마실 수 있도록 한다.

- 1. 포도주 잔을 선택하면 그 포도주를 모두 마시고 되돌려 놓는다.
- 2. 연속으로 놓여 있는 3잔을 마실 수 없다.

#### Facts:

- 포도주 잔의 개수를 저장할 배열
- 각 포도주 잔에 담겨있는 양의 정보
- 다이나믹 프로그래밍을 위한 계산 정보 저장 배열

### OverView:

일반적으로 다이나믹 프로그래밍을 구현하기 위해 낮은 단계 부터의 계산을 실행하고, 그 정보를 저장해 분할 접근과 메모이제이션을 이용한 다이나믹 프로그래밍을 구현한다.

포도잔의 정보 입력받기 -> 반복문을 통해 계산할 수 있는 가장 낮은 수 부터 n까지 계산하며 계산 결과를 저장.

# Algorithm:

전체 포도잔의 개수가 *n개라고* 하면 제시된 문제에서 연속 적으로 마실 수 없는 개수는 '3개' 이므로 다이나믹 프로그래밍의 특성을 이용해 n 번째 전 까지의 계산 결과는 이전 과정에서 완료되어 배열에 저장되어 있다고 가정하고 *n, n-1, n-2* 의 상태에 따른 경우를 분류해 알고리즘을 완성시킨다. (당연히 반복문도 3번쨰부터 시작.)

몇가지 경우를 생각해 보면,

1. **n번째 포도주를 먹었을 경우** -> n-1를 마셨다. -> n-2는 절대 못마신다. -> case 1) -> n-1를 마시지 않았다. -> n-2는 마실 수 있다. -> case2)

2. n번째 포도주를 안먹었을 경우 -> n-1, n-2둘다 상관없다. -> case3)

\*\*\*case1 : n-2를 <mark>제외</mark>해야함 -> n-3까지의 최대값에 n-1, n의 양을 더하면 된다.

case2 : n-1를 마시지 않았고 n-2는 <mark>상관이없음</mark> -> n-2까지의 최대값에 n의 양을 더함

case3: n을 안마셨을 경우 그 전의 n-2, n-1는 <mark>상관이 없음</mark> -> n-1까지의 최대값.

key point: 상관이 없냐 제외를 해야하냐의 조건에 따라서 그때 까지의 최대값을 불러와도 되느냐, 아니면 순수히 양만 더해 줘야 하느냐가 정해짐.

# Time Complexity:

 $n + n \rightarrow O(n)$