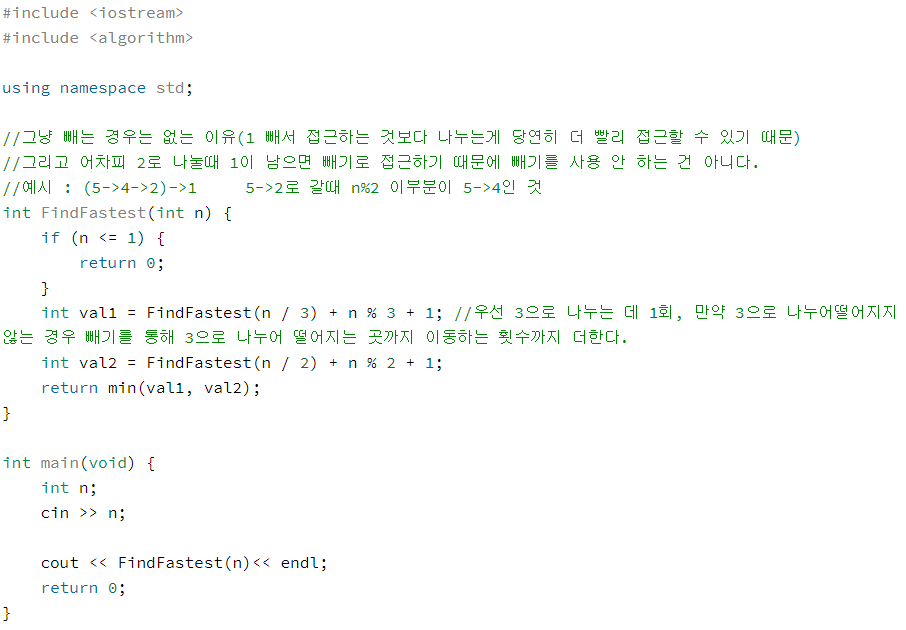
1. 이번 풀이 방법



우선 1을 빼서 접근하는 방법은 2나 3으로 나누는 방법에 비해서 매우 비효율적이다.

예시

6->2

빼기 6->5->4->3->2->1

2or3으로 나누기 6->2->1

따라서 1을 빼는 방법은 일반적으로 고려하지 않고, 2나 3으로 나누어 떨어지지 않는 경우 해당 수가 나누어 떨어지기 위한 수까지 접근하기 위한 방법으로 사용할 것을 고려해야한다. 이 때 1을 빼는 횟수는 2의 경우 n%2, 3의 경우 n%3이다.

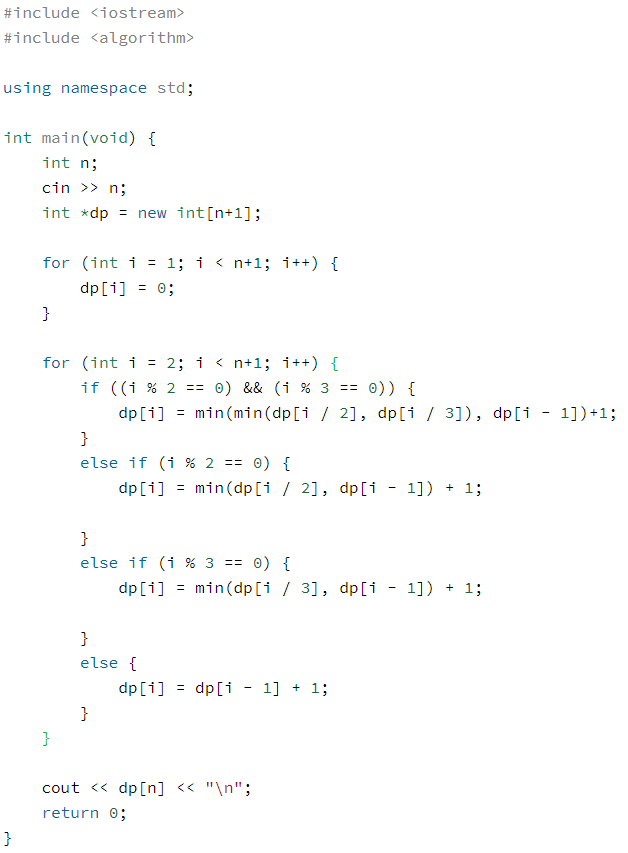
이에 따라 2또는 3으로 나눴을 때 나오는 수를 다시 재귀함수로 불러 해당 수(n/2, n/3)에서의 최솟값에 n/2, n/3까지 접근하기 위해 사용한 나눗셈 1회, 그리고 나누어 떨어지지 않았을 경우 빼기로 접근하기 위한 n%2, n%3을 더한 수 중에서 최솟값이 n에서 1로 가기 위한 최소 횟수이다.

식으로 표현하면

Result=min(func(n/2)+n%2+1,func(n/3)+n%3+1)으로 나타낸다. 그리고 1의 경우에는 굳이 계산할 필요가 없기 때문에 0을 반환하면 된다.



2) 이전 풀이 방법



배열을 사용해서 메모이제이션을 사용하는 방법이다.

배열 dp[]을 동적 할당하여 초기값으로 0을 준다. 이후 숫자 1의 경우 이미 완성이 되어있으므로 dp[1]=0을 주고, 2부터 n까지 반복문을 반복하면서 6, 3, 2의 배수인 경우와 그 외의 수에 대해서 접근 가능한 수를 체크하여 최솟값을 arr[n]에 집어넣는다. 예를 들어 6의 배수인 경우 2와 3으로 나누는 방법, 그리고 1을 빼는 방법이 모두 가능하기 때문에 해당 수의 dp[]값을 찾아 3개의 수중에 최소값에다가 1을 더해(해당 수를 가는 횟수) dp[i]에 집어넣는다.



두 방법의 비교

2부터 n까지 모든 수에 대한 메모이제이션을 진행하고(1번 방법의 경우 숫자가 큰 경우 건너뛰는 수가 많다.) if문으로 2의 배수, 3의 배수, 6의 배수, 그 이외의 수로 나누어 게산하다 보니 계산이 오래 걸린 것 같다.