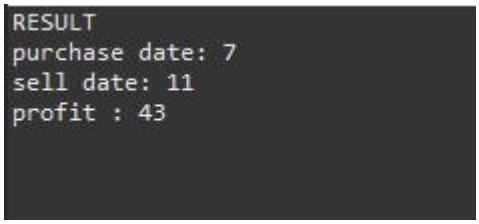
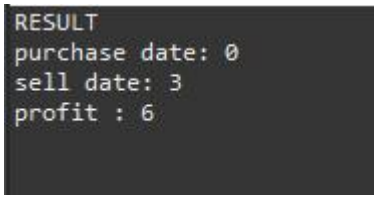
알고리즘 HW04
- dynamic programming-

제출일자	2020.10.7
과제번호	04
이　　름	강인한
학　　번	201701969

실행결과





작성한 코드 시간 복잡도 설명 : 해당 코드의 가장 큰 시간 복잡도를 갖는 것은 반복문 for이다. for 안에 if 문이 들어가 있지만, 상수 시간은 O(n)의 시간에 큰 영향을 주지 못하므로 이 코드의 시간 복잡도는 O(n)이라고 볼 수 있다.

풀이과정 : 날마다 값이 변하는 주식을 가장 많이 이득을 보기 위해 다음날의 가격에서 오늘의 가격을 뺀 값을 갖는 배열을 만들었다. 해당 배열을 바탕으로 최대 부분합을 구한다면 이 부분합은 최대 이득이 된다. dynamic programming 방법으로 최대 부분합을 구하기 위해 반복문을 사용하였다. 배열의 크기만큼 반복하여 커지는 Tabulation 방법을 선택했다. 반복을 통해 1개의 최대 합, 이를 이용한 2개 일 때의 최대 합 이러한 과정을 반복해 나가면 n개일 때의 최대 합 또한 구할 수 있다.

코드 설명 :

```
if (difference[i] >= difference[i] + sum_max) {  // i번째 difference가 더해질게 큰지 작은지
    sum_max = difference[i];  // i번째 index 자체가 크다면  값 저장
    purchase_date = i;  // 구매일 저장
} else {
    sum_max = sum_max + difference[i];  //기존의 것에 i번째 index를 더한값이 크다면 저장
}
```

sum_max와 purchase에 0이 저장되어있는 상태에서 (다음날가격 – 오늘가격)이 저장돼있는 difference 배열의 i번째 index와 sum_max값이 추가된 값 중 큰 값을 sum_max에 저장하여 최신화한다. 최신화된 날(index) 구매한 날이 될 것이므로 저장해 둔다.

위에서 저장된 sum_max의 값과 바뀌기 전의 sum_max 값이 저장된 profit 의 크기를 비교하여 새로운 값이 크다면 새로운 값으로 바꿔주고 기존의 것이 크다면 아무일도 하지 않는다. 이때 기존의 것보다 새로운 값이 크다면 해당 i(=index)는 판매일이 될 것이므로 저장한다.

```
if (sum_max >= profit) {  //기존의 sum_max값과 최신화된 값 저장
    profit = sum_max;
    sell_date = i;  // 기존의 값보다 최신화된 값이 크다면 그날은 판매일이 됨
}
```

위의 두 if문을 i=0에서 배열의 size까지 반복하는 작업을 통해 최대 부분합을 구해낼 수 있다.

과제를 통해 느낀 점 : 동적 프로그래밍 방식은 꽤 자주 사용하는 방식이라고 생각합니다. 기존의 값을 활용하여 비교하여 최신화하는 작업은 비교적 많이 해봐 많이 어렵게 느껴지진 않았습니다. 하지만 for 문 하나만으로 완성하기 위해 간단한 알고리즘을 생각해 내는 것은 어려웠던 것 같습니다.